

 **INFORMATICA**

DORIN IONIȚĂ CÂRSTOIU

SISTEME EXPERT



Sisteme expert

conf. dr. ing. **Dorin Cârstoiu**
Universitatea "Politehnica", București

PREFAȚĂ

Sistemele expert sînt produse ale Inteligenței Artificiale, ramură a științei calculatoarelor ce urmărește dezvoltarea de programe inteligente. Cu toate că existența marchează peste 25 ani, ele devin populare abia în ultimii ani ca urmare a scăderii costurilor informațiilor tehnologice și a calculatoarelor, motiv pentru care numărul utilizărilor potențiali este într-o continuă creștere. Ceea ce este remarcabil pentru sisteme expert, este aria de aplicabilitate ce a cuprins multe domenii de activitate de la arhitectură, bănci, comerț, educație, ingineria sistemelor, etc.

Această carte nu favorizează un domeniu în detrimentul altuia, ea încearcă să surprindă elemente de bază în dezvoltarea sistemelor expert, metodologii de lucru, clase de aplicații. Cartea este structurată în zece capitole ce nu pot fi parcurse independent, fiecare fiind o parte din ansamblul de concură la dezvoltarea unui astfel de sistem.

Capitolul 1 se dorește o introducere în domeniu și totodată o definire a noțiunii de sistem expert. Sînt analizate principalele sisteme expert cunoscute subliniindu-se în principal particularitățile lor.

Capitolul 2 are ca obiectiv introducerea cititorului în teoria limbajelor formale și a logicii matematice, cu accente pe limbajele de ordinul întâi, a teoriei inferențelor logice și logicii vagi.

Capitolul 3 descrie principalele componente ale unui sistem expert, arhitecturi complete, sistem esențial, structuri de sisteme expert destinate activităților ingineresti. O atenție deosebită este acordată sistemelor în timp real, sisteme pentru care timpul este un parametru critic.

Metodele de reprezentare a cunoașterii formează obiectul capitolului 4. Sînt analizate aici atît activitățile privind organizarea, clasificarea și metareprezentarea, cît și principalele moduri de reprezentare a cunoașterii. Întrucît performanțele oricărui sistem expert sînt dependente de conținutul și organizarea bazei de cunoștințe, se au în vedere algoritmi de organizare a acesteia în scopul simplificării strategiei de control.

O mare atenție este acordată sistemului rezolutiv. Acesta face obiectul mai multor capitole, capitolul 5 tratînd problematica generală a sistemului

O mare atenție este acordată sistemului rezolutiv. Acesta face obiectul mai multor capitole, capitolul 5 tratând problematica generală a sistemului rezolutiv, tipuri de strategii de control, reprezentarea problemelor, metoda rezoluției, reprezentarea în spațiul stărilor. Capitolul 6 este destinat strategiilor de căutare și tratează în principal metode de căutare în arbori și grafuri. Pentru fiecare metodă de căutare sînt prezentați în pseudocod algoritmi utilizați, se dau demonstrații de consistență și convergență a acestora. În capitolul 7 se tratează metodele de rezolvare a problemelor prin decompoziție, avînd ca principal obiectiv tratarea metodelor de căutare în grafuri ȘI/SAU. Controlul rezolvării problemelor, utilizarea experienței, planuri ierarhizate fac obiectul capitolului 8. Pentru susținerea rezultatelor teoretice un mare număr de exemple este prezentat.

Capitolul 9 încearcă să surprindă categoriile de aplicații pentru a căror rezolvare sînt necesare sisteme expert, cît și fazele și metodologia de realizare a acestora.

În final, sînt prezentate trei aplicații simple, dar cu înalt grad de generalitate ce nu utilizează un sistem cadru pentru dezvoltare. Sursele programelor scrise sînt prezentate în speranța ca cititorii vor fi mai mult atrași de acest domeniu interdisciplinar și de mare actualitate.

Aș dori să mulțumesc pe această cale colectivului de cadre didactice al facultății de Automatică și Calculatoare, care a avut un rol important în formarea mea profesională. Vii mulțumiri le sînt adresate studenților mei ce au contribuit nemijlocit la definitivarea materialului prin discuțiile purtate. În mod special aș dori să evidențiez absolvenții Hagi Daniela, Anghel Alice și Papacica Adrian care au contribuit la realizarea aplicațiilor prezentate în capitolul 10.

Lucrarea este destinată unei categorii largi de cititori, indiferent dacă sînt specialiști în calculatoare, întrucît problemele și tehnicile Inteligenței Artificiale și în special ale Sistemelor Expert au fascinat un mare număr de cercetători din domenii disciplinare foarte variate. Textul poate fi utilizat și de material pentru cursul Sisteme Expert în Automatica, predat studenților din anul IV ai facultății de Automatică și Calculatoare.

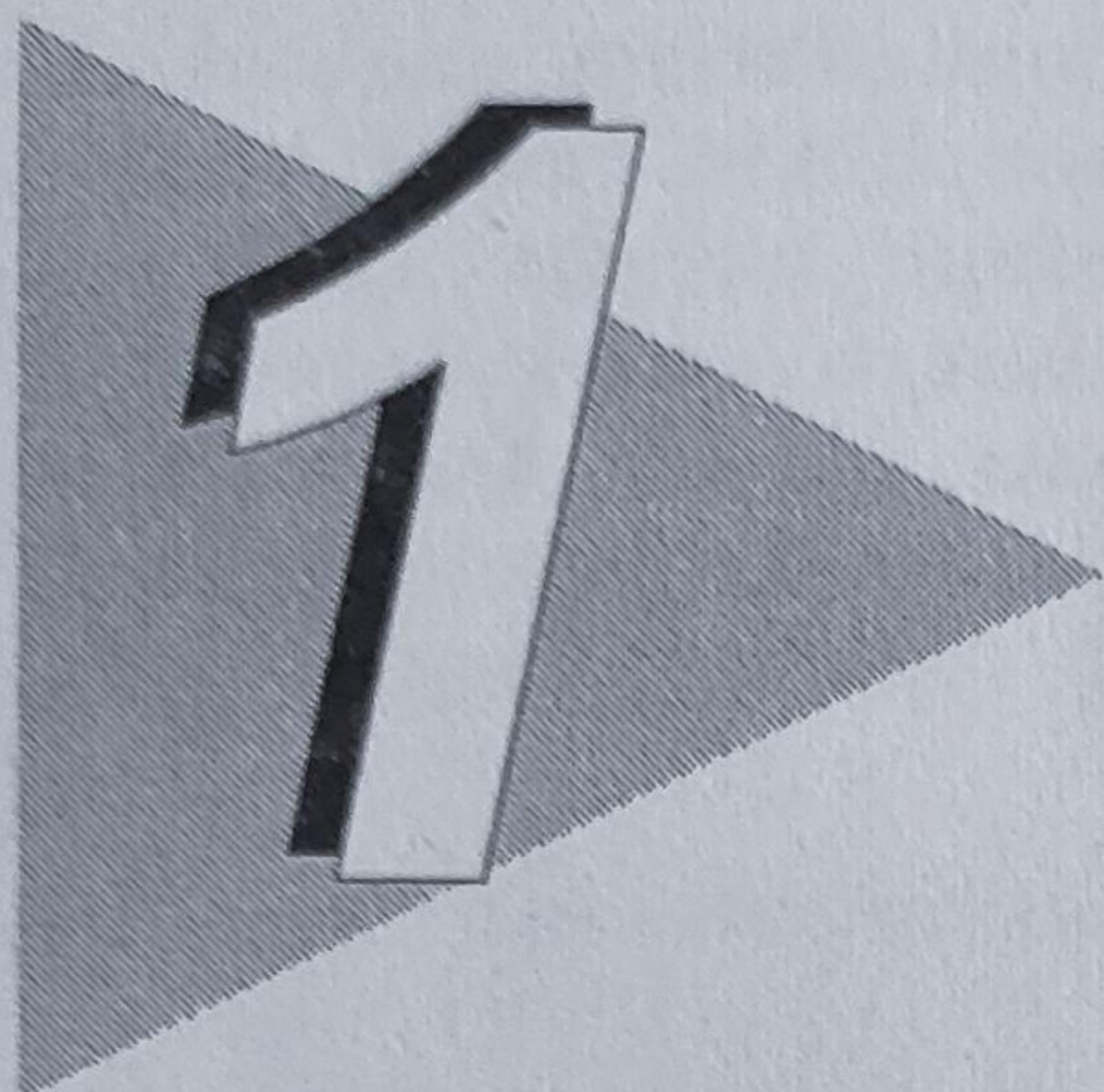
CUPRINS

1. INTRODUCERE	1
1.1. ISTORIA INTELIGENȚEI ARTIFICIALE	3
1.2. SISTEME EXPERT, DEFINIȚII	6
1.3. ORIGINEA SISTEMELOR EXPERT	10
1.4. CARACTERISTICI ALE SISTEMELOR EXPERT CU APLICAȚII IN TIMP REAL.....	13
2. METODE MATEMATICE ÎN INTELIGENȚA ARTIFICIALĂ ȘI SISTEME EXPERT.....	17
2.1. LIMBAJE FORMALE, LIMBAJE DE ORDINUL INTII	19
2.2. LIMBAJUL CALCULULUI PROPOZIȚIONAL	25
2.3. LIMBAJUL CALCULULUI CU PREDICATE DE ORDINUL INTII	29
2.4. TEORIA INFERENȚELOR LOGICE	32
2.4.1. Reguli de inferență structurală	36
2.4.2. Reguli de inferență logică	36
2.5. TEHNICI FUZZY.....	37
2.5.1. Teoria seturilor fuzzy	37
2.5.2. Logica fuzzy.....	40
2.5.3. Sisteme expert ce utilizează logica fuzzy.....	41
3. ARHITECTURI DE SISTEME EXPERT	43
3.1. COMPONENTELE SISTEM EXPERT	46
3.2. SISTEME EXPERT PENTRU APLICAȚII IN TIMP REAL	51
4. REPREZENTAREA CUNOAȘTERII	53

4.1. NOȚIUNI INTRODUCATIVE DESPRE REPREZENTAREA CUNOAȘTERII	56
4.1.1. Sistemul de meta-reprezentare	57
4.1.2. Sistemul de clasificare	58
4.1.3. Sistemul de organizare	59
4.2. METODE DE REPREZENTAREA CUNOAȘTERII	60
4.2.1. Reprezentarea cunoașterii în limbajul calculului cu predicate de ordinul întâi	64
4.2.2. Metode procedurale de reprezentarea cunoașterii	79
4.2.3. Reprezentarea cunoașterii prin rețele de producție	80
4.2.3.1. Analiza condiției	84
4.2.3.2. Metode de rezolvare a conflictelor	85
4.2.3.3. Transmiterea acțiunii	88
4.2.3.4. Organizarea sistemelor de producție	89
4.2.3.5. Măsurile pentru creșterea performanțelor	89
4.2.3.6. Inconsistența sistemelor de producție	93
4.2.4. Rețele semantice	95
4.2.4.1. Rețele semantice simple	107
4.2.4.2. Rețele semantice sortate	109
4.2.4.3. Rețele semantice extinse	118
4.2.5. Reprezentarea cunoașterii cu ajutorul cadrelor	119
4.2.5.1. Reprezentarea acțiunii în cadre	122
4.2.5.2. Introducerea regulilor în cadre	123
5. SISTEME REZOLUTIVE	125
5.1. CICLUL DE BAZĂ AL UNUI MECANISM DE INFERENȚĂ	128
5.2. STRATEGII DE CONTROL ÎN MECANISME INFERENȚIALE	129
5.2.1. Strategia de control înainte	129
5.2.2. Strategia de control înapoi	133
5.2.3. Strategia de control combinat	134
5.2.4. Strategia de control circumstanțial	135
5.2.5. Alegerea strategiei de control	135
5.3. ANALIZA TIPURILOR DE PROBLEME	136
5.4. PROCESUL REZOLVĂRII ȘI CUNOAȘTEREA	138
5.5. REPREZENTAREA PROBLEMELOR ÎN LIMBAJUL CALCULULUI CU PREDICATE DE ORDINUL ÎNȚII	140
5.5.1. Particularități ale reprezentării problemelor în limbajul calculului cu predicate de ordinul întâi	140

5.5.2. Exemple de reprezentare a problemelor utilizând calculul cu predicate de ordinul întâi.....	141
5.6. SKOLEMIZAREA FORMULELOR	145
5.7. REPREZENTAREA IN SPAȚIUL STĂRIILOR.....	149
5.8. UTILIZAREA GRAFURILOR ȘI/SAU	155
6. STRATEGII DE CONTROL.....	159
6.1. CĂUTAREA EXHAUSTIVĂ (TRIAL-AND-ERROR-SEARCHING).....	161
6.2. CĂUTAREA PRIN EXAMINAREA SEMANTICĂ A NODURILOR	162
6.2.1. Căutarea în adâncime	162
6.2.2. Căutarea pe orizontală (în lărgime).....	164
6.3. CĂUTAREA SOLUȚIEI OPTIME	165
6.4. CĂUTAREA CIND COSTUL PENTRU ATINGEREA OBIECTIVULUI POATE FI PREDICTAT	167
6.4.1. Metoda gradientului (Hill-climbing method)	168
6.4.2. Metoda celei mai bune prime căutări.....	170
6.4.3. Soluția optimă când costul poate fi predictat.....	170
6.4.4. Caracteristici ale algoritmului A^*	173
6.4.5. Îmbunătățiri ale algoritmului A^*	176
6.5. APLICAREA UNEI METODE DE CĂUTARE PRIN RAMIFICARE ȘI LIMITARE	180
7. REZOLVAREA PROBLEMELOR PRIN DECOMPOZIȚIE.....	187
7.1. DESCOMPUNEREA PROBLEMELOR	189
7.2. GRAFURI ȘI/SAU	191
7.3. CĂUTAREA IN GRAFURI ȘI/SAU	193
7.3.1. Evaluarea și expandarea grafurilor candidate	193
7.3.2. Căutarea în adâncime în grafuri ȘI/SAU	194
7.3.3. Căutarea soluției optime în grafuri ȘI/SAU.....	197
7.4. CĂUTAREA IN ARBORI PENTRU JOCURI	200
7.4.1. Arbori de jocuri	200
7.4.2. Metoda minimax	201
7.4.3. Metoda alfa-beta	202
7.4.4. Utilizarea metodei de căutare în grafuri ȘI/SAU	204

8. CONTROLUL REZOLVĂRII PROBLEMELOR	209
8.1. GENERAL PROBLEM SOLVER (GPS).....	211
8.2. PLANURI.....	212
8.2.1. Reprezentarea problemelor și metode de soluționare de bază	212
8.2.2. Utilizarea experienței.....	216
8.2.3. Inferențe cu mai multe obiective	219
8.2.4. Schimbarea planului.....	220
8.2.5. Planuri ierarhizate	224
9. METODOLOGIA CONSTRUIRII SISTEMELOR EXPERT	229
9.1. CATEGORII DE SISTEME EXPERT	231
9.2. ALTERNATIVE IN CONSTRUCȚIA UNUI SISTEM EXPERT	233
9.3. ETAPE ALE REALIZĂRII UNUI SISTEM EXPERT.....	236
10. APLICAȚII	239
10.1. SISTEME EXPERT PENTRU SUPRAVEGHEREA EVOLUȚIEI MĂRIMILOR CONTROLATE.....	241
10.1.1. Structura sistemului.....	242
10.1.2. Reprezentarea cunoștințelor	242
10.1.3. Implementarea sistemului pentru achiziția cunoștințelor	243
10.1.4. Implementarea mecanismului de inferență	255
10.1.5. Integrarea în aplicație	258
10.2. SISTEME EXPERT PENTRU ACORDAREA OPTIMĂ A BUCLELOR DE REGLARE.....	258
10.2.1. Dependente între parametrii și performanțe.....	260
10.2.2. Strategia de ajustare a parametrilor	262
10.2.3. Integrarea unui sistem expert cu aplicații în timp real	263
10.2.4. Implementarea	264
10.3. SISTEM EXPERT PENTRU DIAGNOZA MEDICALĂ.....	276
10.3.1. Implementarea mecanismului de inferență.....	276
10.3.2. Baza de cunoștințe	280
11. BIBLIOGRAFIE	285



INTRODUCERE

Inteligența Artificială este o ramură a științei promovată recent în universități și laboratoare de cercetare. Astfel de tehnici au început să fie promovate cu succes și în industrie. Se apreciază ca în viitor tratarea problemelor ingineresti cu metode specifice inteligenței artificiale va crește considerabil. În plus, tehnicile dezvoltate pe baza programării clasice dau rezultate corespunzătoare când se utilizează în descrierea elementelor finite, la simularea circuitelor, prelucrarea problemelor algoritmice, nefiind adecvate la o serie de probleme ingineresti. Metodele ingineresti pot fi caracterizate după B. V. Koen - 1985 prin "utilizarea algoritmilor euristici în scopul determinării celei mai bune soluții într-o situație concret dată".

1.1. Istoria inteligenței artificiale

Antecedentele IA trebuiesc căutate la vechii greci Gorder 1986, dar ideea formării unei discipline de cercetare a inteligenței umane și a raționării formale a condus diverse grupuri din S.U.A. și Europa după cel de-al doilea război mondial. Cercetările includ pionieratul în cibernetică și teoria calculului. Ideea de bază în cibernetică trebuie să se bazeze pe principiul "reacției negative" Wiener 1948, prin care se controlează modul în care fenomenele fizice pot fi generalizate pentru formarea bazei gândirii umane. Alte argumente pentru necesitatea accesului la comportarea umană este bazată pe luarea deciziilor raționale. În conjuncție cu aceste investigații, devine imposibil de exprimat o formulare a tehnicilor analitice cantitative despre aceste operații de cercetare. Evoluția științei calculatoarelor și comportarea umană a stimulat interesele raționării pe mașină, deoarece nu oferea mijloace ce puteau fi utilizate în modelele de calcul ale gândirii și raționării. Aceasta a determinat o nouă disciplină, cea a inteligenței artificiale. Fără procedurile de calcul, nu se pot ilustra conceptele raționării pe mașină utilizând hârtie, creion și modele fizice. Ca potențialul raționării pe

o mașină să poată fi realizat, este necesară reunirea eforturilor de cercetare din mai multe discipline formale. Prin cercetarea începută în 1956 și cu evenimentul consolidării prin Summer Research Project realizat la Dartmouth College, John McCarthy a stabilit termenul de IA. Primele succese sînt datorate programelor capabile să demonstreze teoreme ale logicii matematice (1963) ce au determinat noua deschidere către programe ce folosesc tehnici de IA și o revigorare a domeniului IA, devenind astfel o disciplină, mai mult decît o căutare matematică pentru mașinile inteligente. În timp, a apărut și ideea benefică a înglobării filozofilor, lingviștilor și neurologilor în aceste domenii.

Se pune întrebarea firească asupra căror zone științifice se extinde IA. Se dau mai jos cîteva exemple:

- ♦ vedere artificială - ce presupune recunoașterea formelor, identic cu vederea umană;
- ♦ robotica - focalizează producerea dispozitivelor mecanice capabile să reproducă mișcarea;
- ♦ prelucrarea vocii - ce privește constituirea și sinteza vocii umane;
- ♦ prelucrarea în limbaj natural - înțelegerea și vorirea în limbaj natural;
- ♦ demonstrarea (producerea) teoremelor - în matematică și logică;
- ♦ "General Problem Solving" - rezolvarea unei clase generale de probleme exprimate în limbaje formale;
- ♦ recunoașterea formelor - recunoașterea și clasificarea diferitelor forme;
- ♦ teoria jocurilor;
- ♦ învățarea automată - mașini ce acumulează cunoștințe prin observarea exemplilor.

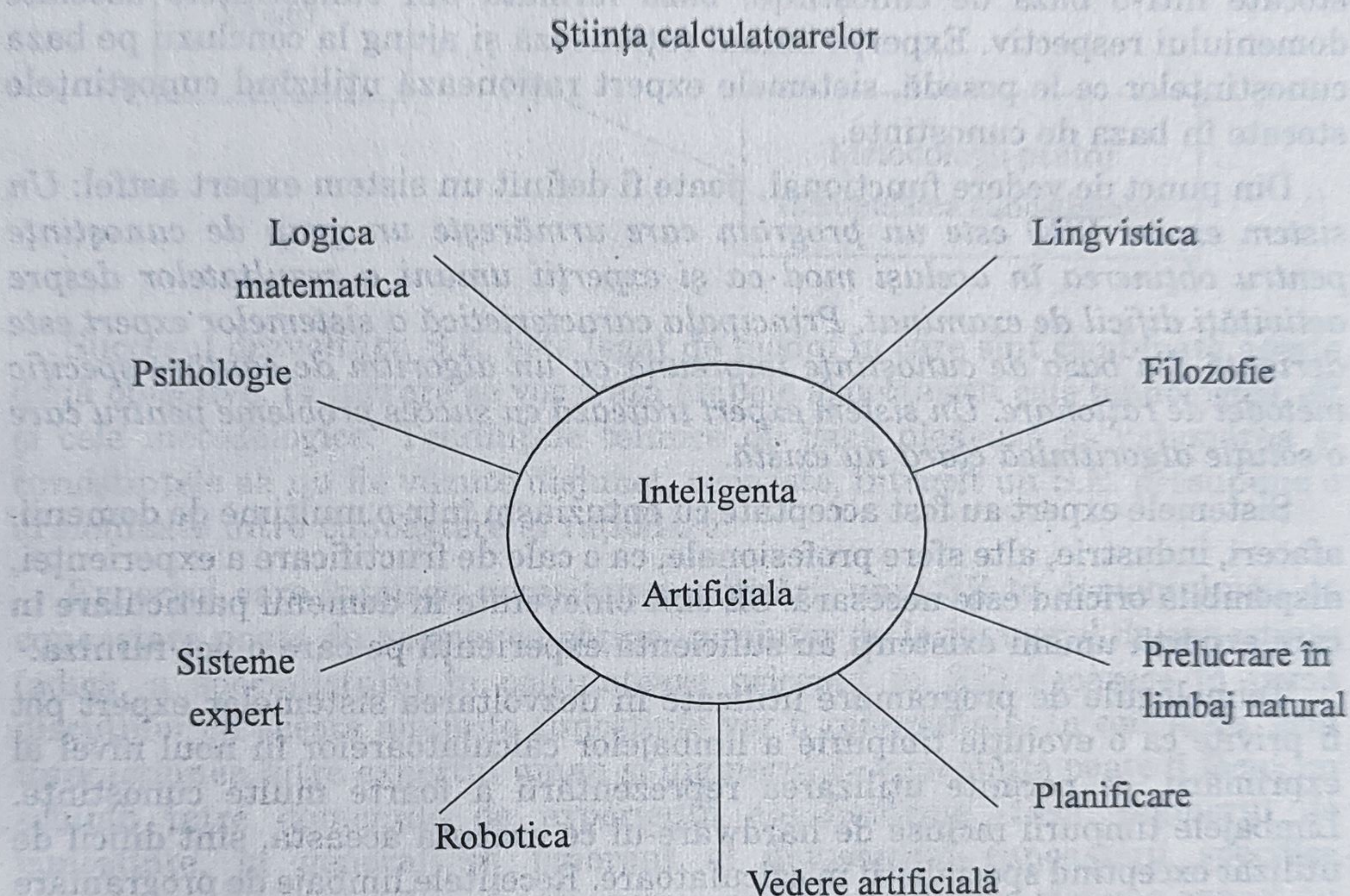
Investigațiile în domenii cum sînt recunoașterea formelor și prelucrarea vocii sînt noi domenii în IA. Nu trebuie să surprindă faptul că utilizarea în ultimii ani a sistemelor expert a fost asociată cu cercetările de IA deoarece structura și proprietățile sînt apropiate de produsele IA.

SE beneficiază de cercetările ce au fost realizate în diferite zone ale IA. În fig. 1.1. se arată relațiile între sistemele expert și celelalte domenii ale IA. În prezent SE sînt tot mai mult asociate direct cu modurile de raționare și cunoștințele reprezentate. SE sînt de fapt dezvoltări în noile cîmpuri ale achiziției de cunoștințe ca forme specializate de învățare, în care cunoștințele sînt achiziționate direct de la expert. Alte SE ca tehnici de IA includ explicații, învățare inteligentă, planificare, rezolvarea problemelor distribuite, cercetări ce sînt adresate direct IA.

Cercetările au fost în general realizate în diferite lucrări pentru doctorate ce au avut mai degrabă scopul demonstrării tehnicilor particulare și a conceptelor, fiind mai puțin preocupate de obținerea unor aplicații particulare.

O mare gamă a cercetărilor urmăresc obținerea minimului în strategii minimax. În problemele privind parcurgerea combinațională a arborilor de căutare, un număr mare de algoritmi au fost dezvoltati pentru o căutare cât mai eficientă.

▼ Fig. 1.1 Domenii ale inteligenței artificiale



Algoritmii se pot grupa după două aspecte:

- ♦ construcția unui arbore de căutare;
- ♦ izolarea și căutarea doar într-o parte a arborelui de căutare.

Cei mai mulți algoritmi de acest tip au caracter euristic. O regulă euristică este o regulă ce oferă o metodă de rezolvare a problemelor, sau o metodă de căutare. Ea nu dă rezultate corecte la orice moment de timp și nu este garantată că găsește cea mai bună soluție, dar în același timp poate reduce timpul de căutare. Metodele de acest tip pot fi folosite pentru reducerea mărimii arborilor de căutare în cazurile arborilor foarte mari. Metodele euristice sînt baza construcției pentru GPS (General Problem Solver).

1.2. Sisteme expert, definiții

O ramură a Inteligenței Artificiale (IA) este reprezentată și de către sistemele expert. Un sistem expert este un program care urmărește cunoștințele, raționează pentru obținerea rezultatelor într-o activitate dificilă întreprinsă uzual doar de experții umani. Dacă un expert uman are cunoștințe într-un domeniu specific, un sistem expert utilizează cunoștințele ce sînt stocate într-o bază de cunoștințe, bază formată din cunoștințele asociate domeniului respectiv. Experții umani raționează și ajung la concluzii pe baza cunoștințelor ce le posedă, sistemele expert raționează utilizînd cunoștințele stocate în baza de cunoștințe.

Din punct de vedere funcțional, poate fi definit un sistem expert astfel: *Un sistem expert (SE) este un program care urmărește un grup de cunoștințe pentru obținerea în același mod ca și experții umani a rezultatelor despre activități dificil de examinat. Principala caracteristică a sistemelor expert este derivată din baza de cunoștințe împreună cu un algoritm de căutare specific metodei de raționare. Un sistem expert tratează cu succes probleme pentru care o soluție algoritmică clară nu există.*

Sistemele expert au fost acceptate cu entuziasm într-o mulțime de domenii: afaceri, industrie, alte sfere profesionale, ca o cale de fructificare a experienței, disponibilă oricînd este necesară. SE sînt binevenite în domenii particulare în care experții umani existenți au suficientă experiență pe care o pot furniza.

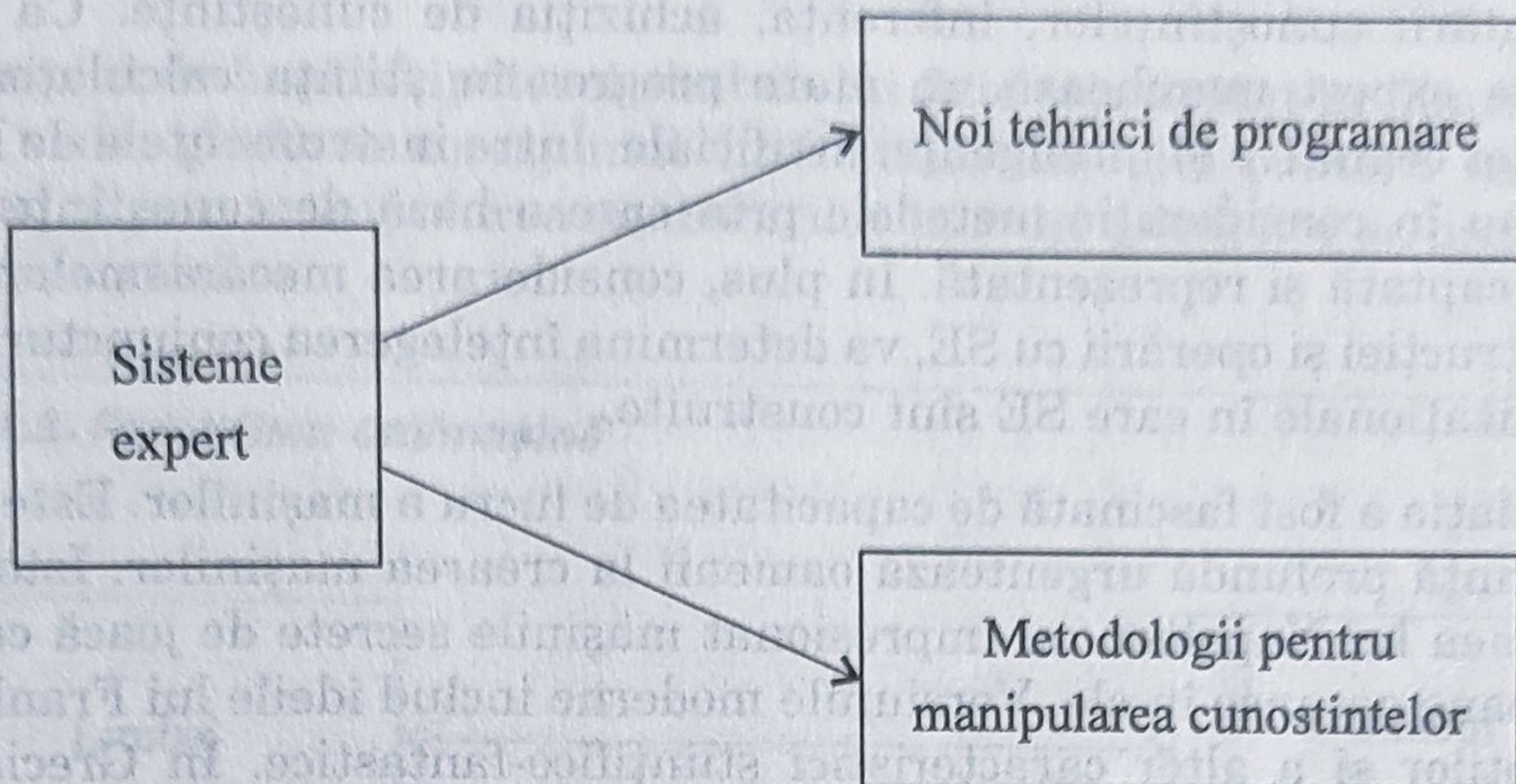
Tehnologiile de programare utilizate în dezvoltarea sistemelor expert pot fi privite ca o evoluție timpurie a limbajelor calculatoarelor în noul nivel al exprimării, ce permite utilizarea reprezentării a foarte multe cunoștințe. Limbajele timpurii incluse de hardware-ul ce executa aceasta, sînt dificil de utilizat exceptînd specialiștii în calculatoare. Recentele limbaje de programare sînt binevenite pentru utilizarea și includerea diverselor forme de expresie umană.

Tehnologiile de programare specifice SE marchează alți pași în această direcție. Pentru a fi utilizată efectiv, fiecare tehnologie nouă este însoțită de un set de ghiduri și metodologii pentru utilizare. Aceasta ajută noii utilizatori la creșterea eficienței utilizării și a maximizării productivității. Din acest motiv utilizarea tehnicilor specifice în SE cu foarte multe cunoștințe va necesita un nou set de metodologii, bazate pe experiența crescută, în dezvoltarea aplicațiilor.

Așa cum se arăta în fig. 1.2 un SE este bazat pe două componente distincte complementare:

- a) noi tehnologii de programare ce permit utilizarea a foarte multe cunoștințe, precum și modul de inferențiere cu acestea;
- b) noi construcții și metodologii dezvoltate, ce permit utilizarea efectivă a acestor tehnologii legate cu probleme complexe.

▼ Fig. 1.2. Componentele de bază ale SE



Succesul dezvoltării S.E. este legat de modul în care sînt combinate aceste două obiective. În lucrare se vor trata ambele aspecte atît cele tehnologice, cît și cele metodologice. Tendințele tehnice de bază pledează ca raționarea și cunoștințele să nu fie văzute disjunct, separate, întrucît un S.E. presupune o armonizare între cunoaștere și raționare.

Expertul care înțelege necesitatea utilizării unui SE în domeniul său de cunoaștere poate de asemenea obține un ajutor de la inginerul de cunoștințe (adică, a specialistului în calculatoare priceput în SE) deoarece în urma discuțiilor cu acesta anumite cunoștințe vor fi reformulate. În concluzie, prin interacțiunea între expertul uman și inginerul de cunoștințe poate fi făcut un schimb între domeniile de experiență ale fiecăruia. Deci, inginerul de cunoștințe, în general un ignorant al domeniului cunoașterii specifice expertului iar expertul cu foarte puține cunoștințe despre sisteme expert au puncte de vedere ce vor fi puse de acord în urma acestei colaborări.

În concluzie, nu este ușor pentru un expert să găsească un inginer de cunoștințe în vederea colaborării deoarece cunoștințele sînt expansive și greu de formulat într-o manieră concisă. Aceasta motivează necesitatea ca experții umani să aibă mai multe cunoștințe despre sisteme expert. Sînt necesare cît mai multe utilitare specifice dezvoltării sistemelor expert, trebuie să fie posibil ca eventual mai mulți experți să fie adepții utilizării SE fără a fi utilizate cunoștințele lor în același timp de către inginerii de cunoștințe. În orice eventualitate punctele de vedere ale inginerului de cunoștințe sînt utilizate în proiectare și este limpede ca procesul construirii sistemelor expert este mai eficient dacă expertul este cunoscător asupra tehnologiilor și metodologiilor sistemelor expert.

Sistemele expert fac apel la o serie de mecanisme de înalt nivel cu dedicație specială fără a se face apel la limbajele clasice cum sînt C și Pascal, mecanismele specifice permițînd implementarea noilor tipuri de exprimări.

Conștientizarea experților despre sistemele expert și cunoașterea inginerescă este de un mare beneficiu pentru dezvoltarea SE. Pentru facilitarea acestora este esențial să se distingă elementele esențiale ale reprezentării cunoștințelor, inferența, achiziția de cunoștințe. Ca urmare, sistemele expert marchează un mare progres în știința calculatoarelor, a sistemelor cognitive și inteligenței artificiale. Între instrumentele de lucru ale SE se iau în considerație metodele prin care o bază de cunoștințe poate fi afișată, captată și reprezentată. În plus, considerarea mecanismelor de bază ale construcției și operării cu SE, va determina înțelegerea conjuncturii sociale și organizaționale în care SE sînt construite.

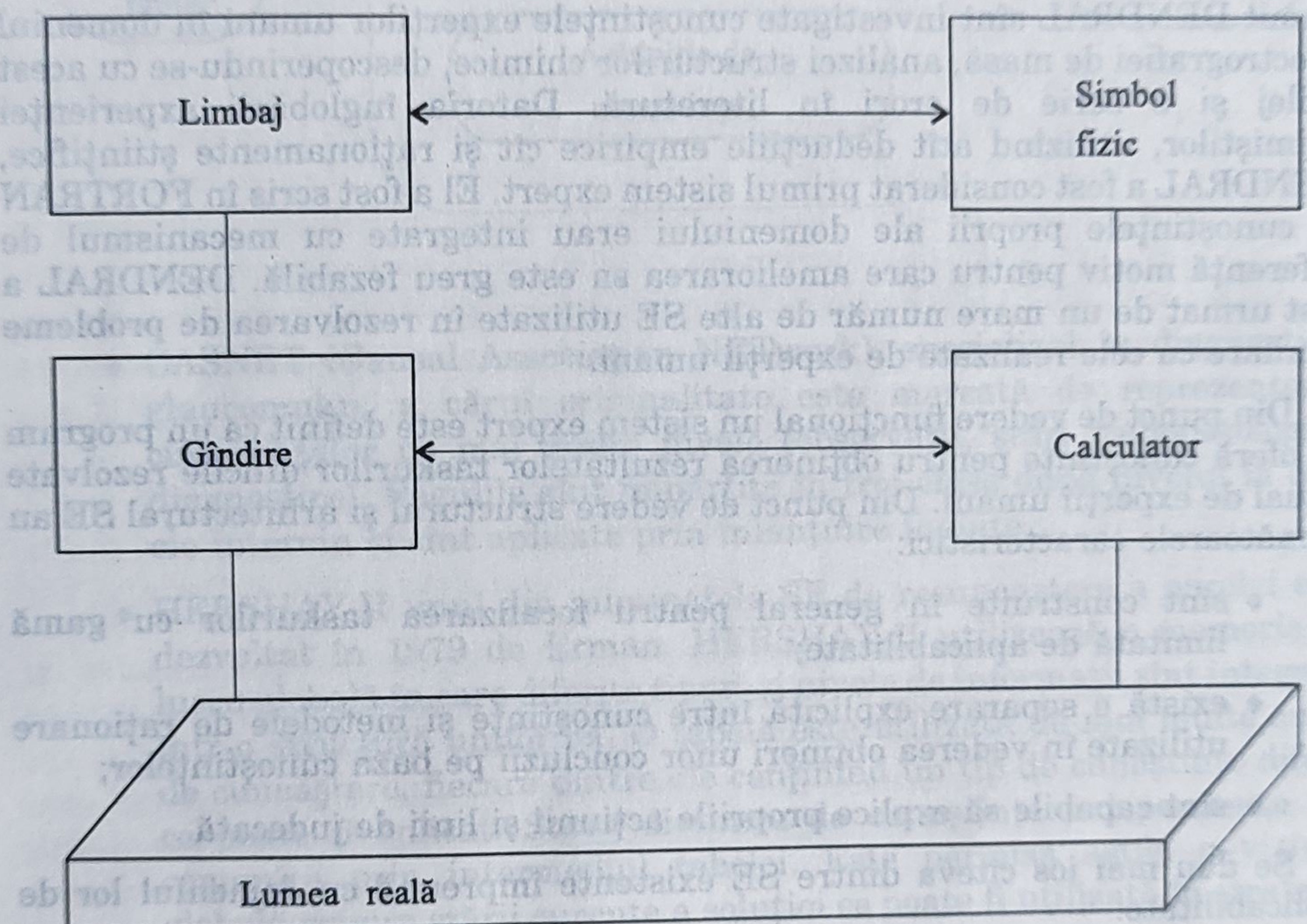
Populația a fost fascinată de capacitatea de lucru a mașinilor. Este evident, ca o dorință profundă urgencează oamenii la crearea mașinilor. Istoric, încă din vremea lui Napoleon au impresionat mașinile secrete de joacă ce operau cu persoane ascunse în ele. Versiunile moderne includ ideile lui Frankenstein ale roboților și a altor caracteristici științifico-fantastice. În Grecia antică filozofii au propus crearea unor mecanisme formale bazate pe logică. Cîteva variante pornesc de la construcția modelelor ce furnizează raționarea mecanizată. Un exemplu este Ramon Lull din Spania, care a construit o mașină ce urmărește demonstrarea faptului că Dumnezeu există. Lull utilizează literele pentru reprezentarea cuvintelor și argumentelor, precum și combinații ale acestor simboluri pe baza unui sistem de reguli. La baza sistemului construit de Lull stă o schemă mecanică de construcție a unei varietăți de figuri geometrice, care rotite unele față de altele determină noi argumente logice. Sistemul său este limitat de stadiul de dezvoltare al geometriei tridimensionale prin numărul operațiilor geometrice posibile. Atingerea scopului propus de SE a fost imposibil de realizat pînă la dezvoltarea calculatoarelor moderne și desăvîrșirii noii revoluții intelectuale. La capacitățile tehnologice actuale mimarea capabilităților umane este o necesitate ca mașinile inteligente să devină o realitate.

Principala virtute a calculatoarelor este abilitatea de realizare a unei mari viteze de calcul. Fiecare limbaj de programare încearcă să facă calculatorul cît mai eficient din punctul de vedere al puterii de calcul. Calculatoarele încep să se extindă la aplicațiile ingineresti și bancare, persoanele interesate de calculatoare încep să utilizeze termenul de sistem de procesare. În plus calculul matematic, sistemele automate, sînt realizate prin intermediul unor programe de calcul ce transpun în mașină gîndirea omului ce le-a programat. Pe măsură ce cercetările evoluează, se dezvoltă așa-zisele programe "elegante". Principalul în aceste activități, este creșterea interesului intelectual, cel comercial rămînînd pe locul secund. Oricînd ne gîndim la programele de interes, trebuie știut faptul că programele de acest tip determină creșterea gîndirii, în consecință, demonstrații interesante au fost construite fără ca aplicațiile să fi fost dezvoltate. Problemele ce se solicită rezolvate prin sisteme expert fac apel la gîndire, la abilitatea de raționare cu alte concepte decît numere, problematica ce devine din ce în ce mai mult o necesitate. Realizarea programelor ce raționează pornește de la faptul că

simbolurile de prelucrare pot fi numere, texte, sau alte concepte. Aceasta duce la "ipoteza simbolului fizic" ca simbol fizic esențial. O mașină ce poate manipula simboluri este și calculatorul.

Principiul de bază al ipotezei simbolului fizic este arătat în figura 1.3. Așa cum se vede, gândirea corespunde limbajului, limbajul poate fi captat prin simboluri și prin manipularea simbolurilor într-un calculator se poate simula procesul de gândire.

▼ Fig. 1.3. Paralelism om/mașină



Pornind de la ipoteza simbolului fizic și cu "bunăvoința construcției mașinilor cu viața", cele mai multe cercetări în inteligența artificială încearcă să rezolve problema reprezentării în limbaje de calcul simbolic. În sprijinul acestui entuziasm și inerentelor dificultăți ale raționării pe mașină au fost realizate o serie de cercetări. O problemă serioasă asupra limitei capacității de memorare și a puterii de calcul se pune în faza inițială a epocii informatice. În urma cercetărilor, utilizând calculatoare din ce în ce mai performante, problema raționării a putut fi rezolvată.

1.3. Originea Sistemelor Expert

După ce au fost trecute limitările de spațiu și timp, apreciate la câteva ordine de mărime, au fost marcate succese în comercializarea mai multor programe ce furnizează taskuri specifice cu cele ale experților umani, numite și sisteme expert. În 1960 NASA a hotărât trimiterea unui vehicul pe Marte, unul dintre scopuri fiind cercetarea structurii chimice a solului acestei planete. Pentru aceasta cercetătorii de la Stanford au construit un program ce a trebuit să înglobeze experiența chimiștilor în identificarea structurii chimice a unor substanțe pornind de la spectrograma de masă. În dezvoltarea programului numit DENDRAL sînt investigate cunoștințele experților umani în domeniul spectrografiei de masă, analizei structurilor chimice, descoperindu-se cu acest prilej și o serie de erori în literatură. Datoria înglobării experienței chimiștilor, utilizînd atît deducțiile empirice cît și raționamente științifice, DENDRAL a fost considerat primul sistem expert. El a fost scris în FORTRAN și cunoștințele proprii ale domeniului erau integrate cu mecanismul de inferență motiv pentru care ameliorarea sa este greu fezabilă. DENDRAL a fost urmat de un mare număr de alte SE utilizate în rezolvarea de probleme similare cu cele realizate de experții umani.

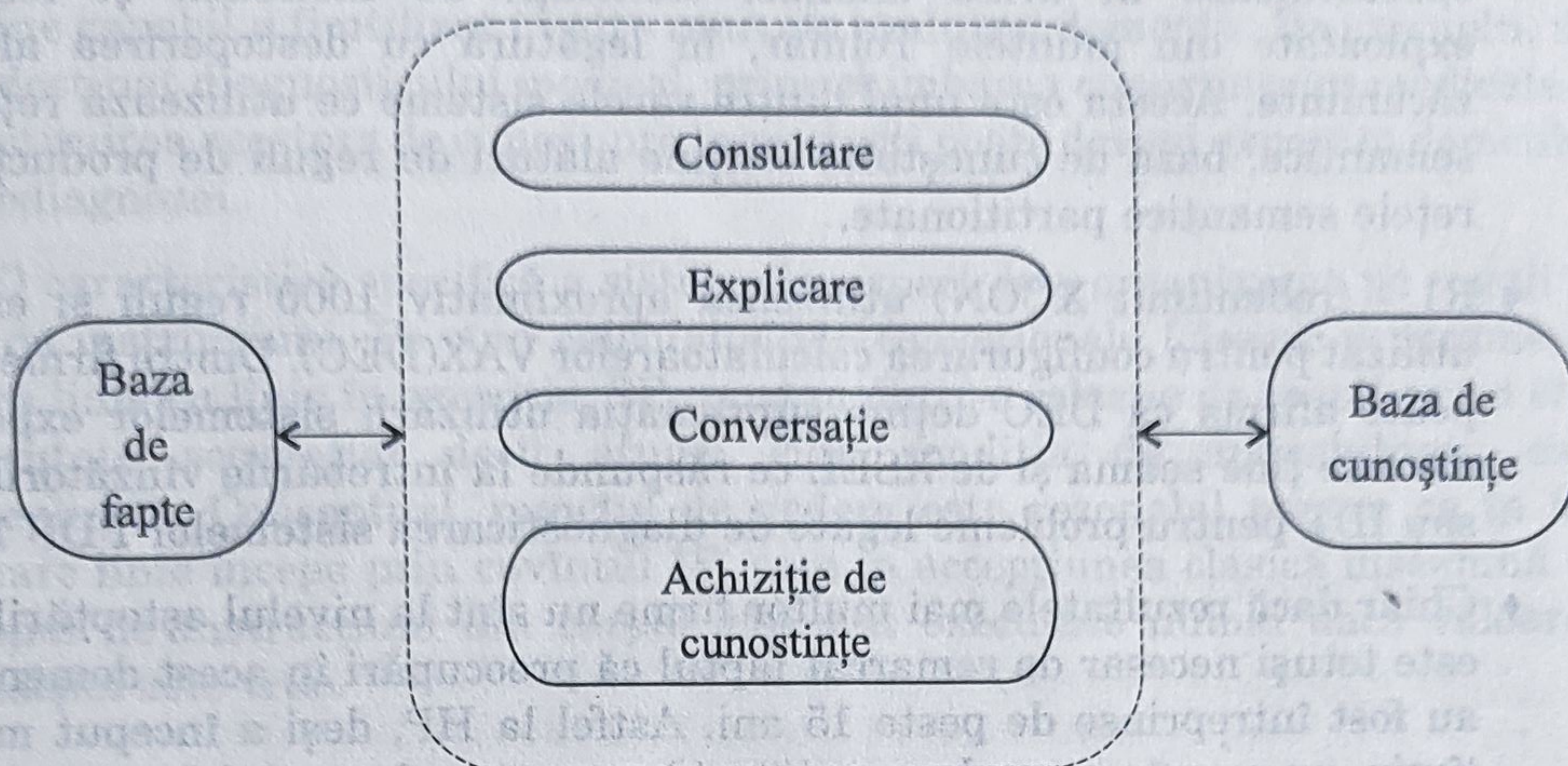
Din punct de vedere funcțional un sistem expert este definit ca un program ce oferă cunoștințe pentru obținerea rezultatelor taskurilor dificile rezolvate uzual de experții umani. Din punct de vedere structural și arhitectural SE au următoarele caracteristici:

- ♦ sînt construite în general pentru focalizarea taskurilor cu gamă limitată de aplicabilitate;
- ♦ există o separare explicită între cunoștințe și metodele de raționare utilizate în vederea obținerii unor concluzii pe baza cunoștințelor;
- ♦ sînt capabile să explice propriile acțiuni și linii de judecată.

Se dau mai jos cîteva dintre SE existente împreună cu domeniul lor de aplicabilitate:

- ♦ MYCIN folosește experiența din DENDRAL și operează ca un consultant pentru medici în investigarea pacienților pentru depistarea infecțiilor bacterigene ale sîngelui și pentru prescrierea tratamentului adecvat. Sistemul operează într-o manieră interactivă, mediul oferind toate informațiile utile despre pacient, cît și rezultatele de laborator. Raționarea în MYCIN în absența informațiilor complete (rezultatele de cultură nu sînt cunoscute decît după 24 pînă la 48 ore) se bazează pe un ansamblu de cunoștințe. El este capabil să-și explice raționamentul, baza de cunoștințe poate fi schimbată la orice moment de timp. MYCIN este compus dintr-o bază de cunoștințe ce cuprindea 300 de reguli în '77, 450 în '80 și este compus din patru secțiuni de program principale ce formează un tot unitar (fig. 1.4). El a fost scris în interlisp pe DEC PDP10 și ocupă 130 K. Dintre aceștia 50 K sînt ocupați de sistemul de consultare, 20 K de sistemul de achiziție a cunoștințelor.

▼ Fig. 1.4. Structura sistemului MYCIN



- ♦ CASNET (Causal Association NETwork), specializat în diagnosticul glaucomului, a cărei originalitate este marcată de reprezentarea cunoștințelor pe mai multe nivele (observații, stări psihopatologice, diagnostice). Regulele sînt împărțite în trei clase după nivelul la care ele intervin și sînt aplicate prin înlănțuire înainte.
- ♦ HERSHAY-II unul din minunatele SE de recunoaștere a parolei este dezvoltat în 1979 de Erman. HERSHAY-II utilizează o memorie de lucru globală în care diferite tipuri și nivele de informații sînt integrate într-o structură uniformă. O tabelă este utilizată de mai multe surse de cunoaștere, fiecare dintre ele conținînd un tip de cunoștințe diferit ce poate fi aplicat. Surse distincte de cunoștințe independente pot comunica prin intermediul tablei. Este permisă astfel o viziune globală asupra stării curente a soluției ce poate fi utilizată în strategia de control.
- ♦ INTERNIST devenit apoi CADUCEUS, dezvoltat la universitatea din Pittsburgh, acoperea în anul apariției (1982) 80% din medicina internă. El cuprinde mai multe milioane de reguli de producție grupate în subiecte particulare. Fiecare boală este legată de observații prin asociativitate în două sensuri, observație - diagnostic și invers, ponderate printr-un coeficient exprimînd frecvența fiecărei observații pentru un diagnostic. Proiectul a fost dezvoltat în mai mult de zece ani într-un colectiv mare de specialiști.
- ♦ MOLGEN, dezvoltat la universitatea Stanford, specializat în biologie moleculară. Dificultatea principală a sistemului a fost legată de necesitatea integrării în baza de cunoștințe a informațiilor diverse de biologie, topologie, chimie, tehnologia manipulării și genetică.

- ♦ PROSPECTOR, dezvoltat la universitatea Stanford, este specializat în prospecțiunile miniere. Prin program au fost obținute rezultate spectaculoase în urma analizei cantității de molibden și fosfor exploatate din muntele Tolmar, în legătură cu descoperirea altor zăcămintele. Acesta este unul dintre rarele sisteme ce utilizează rețele semantice, baza de cunoștințe conține alături de reguli de producție, rețele semantice partitionate.
- ♦ R1 - (redenumit XCON) utilizează aproximativ 1000 reguli și este utilizat pentru configurarea calculatoarelor VAX(DEC). Dintre firme se poate afirma că DEC deține supremația utilizării sistemelor expert dacă se ține seama și de XSEL ce răspunde la întrebările vânzătorilor sau IDT pentru probleme legate de diagnosticarea sistemelor PDP 11.
- ♦ Chiar dacă rezultatele mai multor firme nu sînt la nivelul așteptărilor este totuși necesar de remarcat faptul că preocupări în acest domeniu au fost întreprinse de peste 15 ani. Astfel la HP, deși a început mai tîrziu, un număr mare de cercetători lucrează în domeniul IA avînd în studiu tehnologii de programare, dezvoltări de pachete software pentru aplicații de SE (în principal diagnoza). Honeywell s-a angajat recent în rezolvarea unor chestiuni legate de achiziția cunoștințelor, utilizarea cunoștințelor, arhitecturi și ustensile pentru dezvoltarea SE. IBM și-a orientat cercetările pe diferite sisteme generice capabile să fie introduse în diverse aplicații. IBM dezvoltă mai multe generații de SE destinate diagnosticului produselor informatice. Se remarcă aici și alte firme de renume ca: General Electric, Schlumberger, Litho et Mirlitho, etc.

Dezvoltarea SE cere noi specialiști pentru captarea cunoștințelor și exprimarea acestora sub formă de reguli. Devine necesară o nouă specializare numită "inginer de cunoștințe", un intermediar între specialistul în calculatoare și expertul uman. Metodologia de captare a cunoștințelor este cunoscută sub numele de ingineria cunoașterii. Ca urmare se poate concluziona:

- ♦ un SE poate fi expert într-o singură chestiune. Dacă se construiește un SE ce rezolvă două probleme sînt posibile critici pentru fiecare dintre acestea;
- ♦ un SE poate să rezolve numai acele probleme pe care expertul uman le-a transmis. Dacă se construiește un SE fără a fi consultat expertul uman acesta are cunoștințe limitate și deci este mai slab decît expertul uman;
- ♦ un SE nu va putea niciodată să înlocuiască omul.

O caracteristică specifică SE este marcată de faptul că acestea pot fi adaptive - caracteristică marcată prin faptul că schimbările efectuate la diferite intervale de timp nu determină modificări ale programelor. Pentru aceasta cunoștințele sînt ținute sub formă de reguli care pot fi ușor citite și

modificate de utilizator cu mici intervenții din partea acestuia. În multe SE uzuale există o distincție netă între cunoștințe și mecanisme ce manipulează aceste cunoștințe. Prin modificarea naturii cunoștințelor utilizate, un SE devine capabil a fi utilizat în alte zone ale aceluiași domeniu. De exemplu, un SE destinat diagnosticului medical, prin schimbarea cunoștințelor medicale și substituirea acestora de cunoștințe ingineresti poate deveni expert în domeniul autodiagnozei.

O caracteristică specifică a sistemelor expert este organizarea pe reguli în loc de instrucțiuni. Pe când calculatoarele convenționale folosesc programe ce intră linie cu linie în execuție, SE constau dintr-o colecție de reguli ce nu sînt executate secvențial decît atunci cînd condiția de aplicabilitate este îndeplinită. Conceptual, punctul de vedere este rezonabil pentru ca în SE fiecare linie începe prin cuvîntul IF, care în accepțiunea clasică înseamnă că grupul de instrucțiuni din corpul său sînt executate numai dacă valoarea condiției este true.

1.4. Caracteristici ale Sistemelor Expert cu aplicații în timp real

Dacă sistemul expert rezultat este utilizat în domeniul conducerii proceselor, Astrom și Anton (1984) denumesc noua zonă de aplicații "Expert Control". Astfel, în acest domeniu se distinge un prim nivel de utilizare a unui SE la nivelul unei bucle de reglare într-un controller. El nu este numai o ustensilă pentru utilizator ci este utilizat și în acțiuni directe asupra instalației controlate. În această situație (bucula unitară), universul cunoașterii este limitat. Bineînțeles că la sisteme multivariabile sau pentru analiza globală a performanțelor la nivelul unei instalații un sistem într-o astfel de abordare este recomandat. Trebuie făcută aici o distincție netă între zona teoretică a cercetărilor, focalizate în obținerea algoritmilor de conducere evaluați, și practica inginerască în care algoritmi clasici sînt utilizați și mai ales metodele euristice bazate pe experiența de exploatare a tehnologiilor.

O idee de utilizare a unui SE la nivelul unei bucle de reglare pornește de la sarcina acestui task de armonizare a algoritmilor de aplicație numerici la procesul fizic. Algoritmii numerici pot fi împărțiți în general în trei grupe: algoritmi de reglare, algoritmi de identificare, algoritmi de monitorizare a celor din primele două categorii. Algoritmii de reglare pot fi de complexitate variabilă de la simplii PI, PID la algoritmii optimați. Algoritmii de identificare pot fi simpli algoritmi de estimare sau algoritmi complecși bazați pe metoda celor mai mici pătrate. Supervizarea se referă la detectarea erorilor staționare, depășirea limitelor de alarmare, instabilitate. SE va decide în ce ordine intră în execuție acești algoritmi, ce parametri și la ce valori să fie setați. Rezultatul aplicării unui algoritm sporește cunoașterea în jurul procesului fizic condus și

afectează modul de realizare în viitor a problemei conducerii. În aceste condiții unul din efectele aplicării acestei strategii este creșterea gradului de cunoaștere asupra procesului.

Combinarea SE, algoritmi numerici pentru reglare impune mai multe particularități dintre care se pot enumera:

- ▶ divizarea execuției: algoritmi, logica secvențială de execuție;
- ▶ modul de scriere a rutinelor ce implementează algoritmi numerici va fi adecvată pentru a putea fi executați în paralel cu sistemul expert;
- ▶ rentabilizarea codului fapt ce va simplifica dezvoltările programelor în vedere modernizării;
- ▶ rularea personalului operator de proces, cât și creșterea transportabilității programelor;
- ▶ combinarea mai multor algoritmi de control, posibilitatea de modificare a perioadei de eșantionare și de utilizare a algoritmilor adaptivi;
- ▶ schimbarea într-o manieră facilă a interfeței utilizator;
- ▶ posibilitatea explorării adecvate a informațiilor primare disponibile în jurul părții fixate.

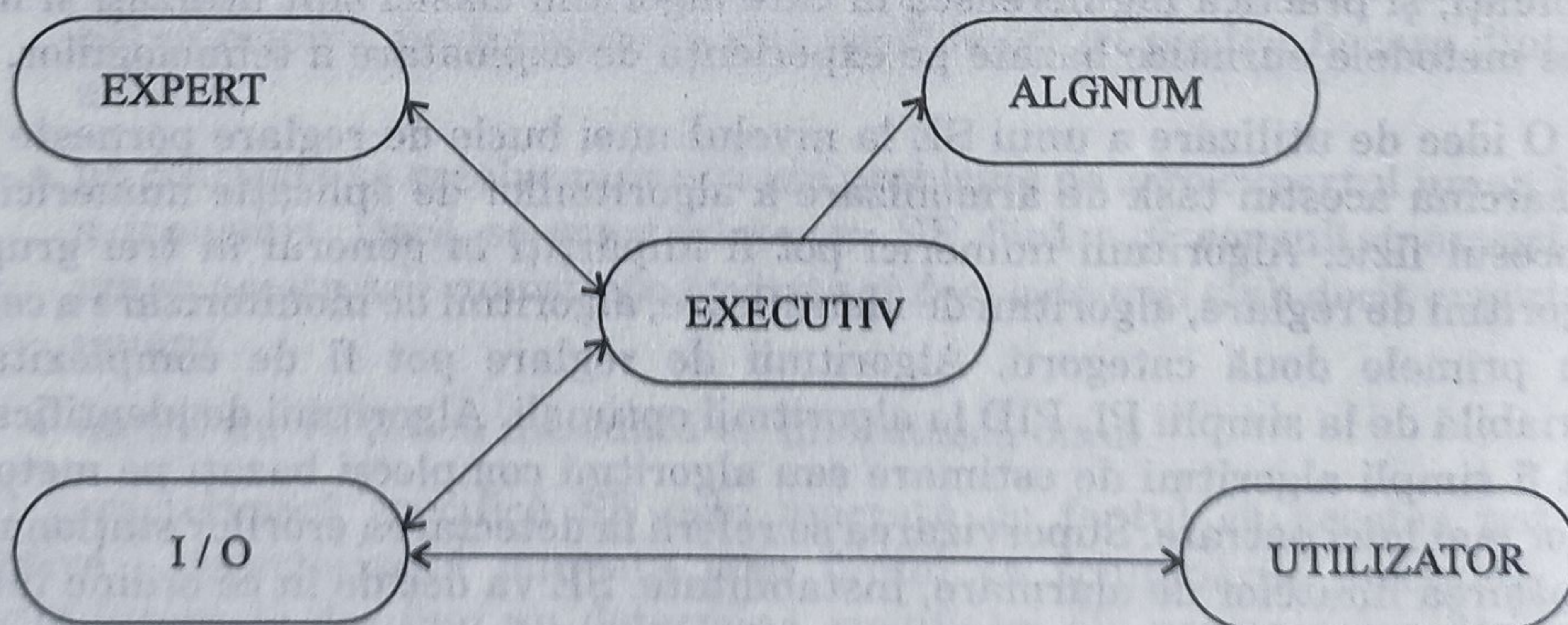
Algoritmi numerici și sistemul expert se vor implementa ca procese paralele separate, ținând cont de la faptul că timpii de rulare sînt diferiți. Dacă timpul de execuție al algoritmului numeric este dependent de partea fixată timpul de răspuns al sistemului expert poate să fie cu cîteva ordine de mărime mai mare. Se consideră pentru exemplificare o structură cu trei procese paralele denumite (fig. 1.5):

EXPERT -partea specifică de sistem expert;

ALGNUM -biblioteca de algoritmi numerici de conducere;

COM I/O -pachet programe pentru comunicație cu utilizatorul.

▼ Fig. 1.5. Structura unui SE la nivelul unei bucle de reglare



Singurul proces critic din cele descrise este ALGNUM, motiv pentru care este organizat ca task cu prioritate maximă. Cele trei procese comunică prin mecanisme de tip cutie poștală, în care mesajele sînt puse într-o coadă din care se poate citi și în care se poate scrie. La încercarea de citire dintr-o cutie goală se va genera un apel de oprire a procesului pînă cînd mesajul sosește. ALGNUM este un proces constituit dintr-o bibliotecă de algoritmi numerici de reglare, estimare și supervizare. Algoritmii sînt asociați cu proceduri de inițializare, execuție, oprire și modificare a parametrilor. Se dau mai jos exemple de mesaje ce pot fi adresate algoritmilor:

- ▶ START (algoritm, parametrii inițiali);
- ▶ PAR (algoritm, nume parametrii, valori);
- ▶ STOP (algoritm);
- ▶ EXECUTA (algoritm); directiva ce este aferentă numai algoritmilor ce nu sînt rulați periodic.

Este posibilă și schimbarea unor variabile globale pentru algoritmii de tip numeric, cum sînt: perioada de eșantionare, referința, mărimea de comandă. O directivă de acest tip este GLOBAL (nume parametrii, valoare).

Alte mesaje sînt transmise de la ALGNUM la EXPERT, și au o prioritate dependentă de semnificație. Mesajele ce provin de la algoritmul de supervizare și cele declanșate de nivelele de alarmare au prioritate mai mare față de cele transmise de algoritmii de estimare. În mod normal informațiile circulă între instalația tehnologică și ALGNUM, sistemul expert nefiind implicat atîta timp cît semnificația a fost bine detectată de algoritmul în lucru. Informațiile primare extrase de către sistemul expert au ca sursă mesajele primite de la ALGNUM. O altă categorie de informații au ca sursă comenzile introduse de către utilizator, comenzi ce semnifică comutarea modului de operare, schimbarea parametrilor, adăugare, ștergere sau editare programe on-line.

2

METODE MATEMATICE ÎN INTELIGENȚA ARTIFICIALĂ ȘI SISTEME EXPERT

2.1. Limbaje formale, limbaje de ordinul întâi

Gradul matematic al sistemelor automate a crescut la nevoie. Pentru a
formula a proceduri de prelucrare a informației de către calculator
Pentru a se face cu ajutorul unor algoritmi implementați pe calculator
succesive de operații care să conducă la un anumit rezultat. Pentru a
informațiilor cu ajutorul calculatorului este necesar ca acestea să fie
de către calculator, pentru care se stabilește o anumită structură de
legată de formele limbajelor formale. Limbajele formale sunt limbaje
care au o sintaxă bine definită și o semnificație precisă. Ele sunt
utilizate pentru a descrie problemele și soluțiile acestora. În
matematică, limbajele formale sunt utilizate pentru a descrie
structurile matematice și relațiile dintre acestea. În informatică,
ele sunt utilizate pentru a descrie algoritmi și structuri de date.
Limbajele formale sunt esențiale pentru dezvoltarea sistemelor
automate, deoarece permit o descriere precisă și neambiguă a
problemei și a soluției acesteia.

646.595



Este rațional a începe cu presupunerea că fenomenele din natură și societate se desfășoară după legi parțial cunoscute. Se pornește de la faptul că regularitatea fenomenelor observate și caracteristicile structurale ale acestora pot determina ordinea conceptuală, ordine ce permite abstractizarea regulilor și legităților.

2.1. Limbaje formale, limbaje de ordinul întâi

Studiul matematic al sistemelor automate a dus la necesitatea descrierii formale a procesului de prelucrare a informației de către calculatoare. Descrierea se face cu ajutorul unor algoritmi, înțelegând prin aceștia o succesiune de operații ce conduc la un anumit rezultat. Pentru prelucrarea informațiilor cu ajutorul calculatoarelor este necesar ca omul să se facă înțeles de către acestea, motiv pentru care el trebuie să studieze toate problemele legate de teoria limbajelor formale, întrucât limbajele de programare nu sînt altceva decît limbaje formale. Intuitiv prin limbaj formal se înțelege o mulțime de șiruri de elemente dintr-un alfabet de simboluri primitive. Aceste șiruri sînt numite în mod obișnuit cuvinte sau propoziții corecte din punct de vedere gramatical. Pentru a selecta cuvintele unui limbaj se mai consideră o mulțime finită de reguli de construcție numită și gramatică. În acest mod problema definirii unui anumit limbaj formal se transpune în problema definirii regulilor gramaticii care generează acest limbaj și a indicării modului în care regulile gramaticii selectează cuvintele limbajului respectiv.

O abstractizare fundamentală este și tipul de similaritate. Se poate defini "tipul de similaritate" ca un 5-uplu cu structura

$$\sigma = \langle I, J, K, \theta, \psi \rangle$$

în care:

I - reprezintă mulțimea de indexare pentru familiile de relații;
 J - mulțimea de indexare pentru familiile de funcții;
 K - mulțimea de indexare pentru elemente distincte;
 $\theta: I \rightarrow N$ - aplicația de aritate a familiilor de relații;
 $\psi: J \rightarrow N$ - aplicația de aritate a familiilor de funcții, în care N este mulțimea numerelor naturale.

Utilizînd conceptul de similaritate se pot clasifica structurile, astfel ca o structură \mathcal{Q}_α avînd tipul de similaritate σ este un cvadruplu cu $i \in I, j \in J, k \in K$ format din:

- ▶ o mulțime A denumită și suportul sau universul lui \mathcal{Q}_α ;
- ▶ o familie $\{R_i^\alpha | i \in I\}$ astfel că pentru fiecare $i \in I$, R_i^α este o relație pe A cu aritatea $\theta(i)$;
- ▶ o familie $\{F_j^\alpha | j \in J\}$ astfel că pentru fiecare $j \in J$, F_j^α este o funcție în A cu aritatea $\psi(j)$;
- ▶ o submulțime $(e_k^\alpha | k \in K)$ a lui A , denumită și mulțimea elementelor distincte din A .

În ansamblu dacă $I = \emptyset$, adică structura α nu conține simboluri relaționale, atunci ea se numește și algebră. Dacă $J = \emptyset$, adică structura α nu conține simboluri funcționale, ea se numește sistem relațional.

Fie ca exemplu structura

$$\alpha = \langle N, \leq, +, *, s, 0 \rangle$$

ce este o aritmetică a numerelor naturale în care:

- \leq - simbolul relațional;
- $+, *$ - simbolurile funcționale binare;
- s - simbolul funcțional numit succesor, $s(x) = x + 1$;
- 0 - simbolul elementului distinct 0, o constantă (constanta zero).

Un "limbaj de ordinul întâi" \mathcal{L}_σ se poate asocia la orice tip de similaritate, astfel că orice enunț din \mathcal{L}_σ , are o valoare de adevăr definită în orice structură avînd tipul de similaritate σ . Un limbaj de ordinul întâi \mathcal{L}_σ este construit din: simboluri primitive, termeni, formule atomice, formule, enunțuri. Din categoria simbolurilor primitive fac parte:

- a) variabilele individuale;
- b) conectivele logice: conjuncție, negație.

Observații. Desigur că se pot utiliza și alte conective logice cum sînt disjuncția și negația $(A \wedge B \rightarrow \neg(\neg A \vee \neg B))$, sau disjuncție, implicație, echivalență ce pot fi obținute din conjuncție și negație.

$$A \vee B \Rightarrow \neg(\neg A \wedge \neg B)$$

$$A \rightarrow B \Rightarrow \neg A \vee B$$

$$A \equiv B \Rightarrow (A \rightarrow B) \wedge (B \rightarrow A)$$

Pot fi reduse conectivele logice necesare și suficiente pentru definirea limbajului de ordin întâi la o singură conectivă "|" definită și conectiva de incompatibilitate, non-conjuncția sau functorul Sheffer. O expresie $A|B$ semnifică faptul că A și B nu sînt ambele adevărate, așa că se pot transcrie conectivele logice uzuale ca în exemplele de mai jos:

$$\neg A \Rightarrow A|A$$

$$A \vee B \Rightarrow (A|A)|(B|B)$$

$$A \wedge B \Rightarrow (A|B)|(A|B)$$

$$A \rightarrow B \Rightarrow A|(B|B)$$

Întrucît reducerea numărului conectivelor logice duce la pierderea semnificației naturale a proceselor inferențiale, pentru rațiuni practice se acceptă drept primitive toate conectivele logice uzuale ale calculului propozițional:

- c) cuantificatorul existențial \exists . Cuantificatorul universal \forall poate fi exprimat cu ajutorul cuantificatorului existențial prin transcrierea

$$\forall xF \Leftrightarrow \neg(\exists x)\neg F$$

- d) simbolurile relaționale $\theta(i)$, $i \in I$;

- e) simbolurile funcționale $\psi(j)$, $j \in J$;

- f) cîte o constantă individuală e_k pentru fiecare $k \in K$.

Se remarcă faptul că unii autori includ printre simbolurile primitive și pe cel de egalitate. În această situație se introduce o regulă suplimentară pentru formarea formelor atomice (adică dacă $E1$ și $E2$ sînt termeni atunci $E1 = E2$ este o formulă atomică) și limbajele se numesc "limbaje de ordinul întâi cu egalitate". După alți autori egalitatea este un simbol relațional și deci nu reprezintă o categorie specială a simbolurilor primitive. Alți autori includ în familia simbolurilor primitive și clasa simbolurilor delimitatoare formată din paranteze, semne de punctuație.

Așa cum cuvintele într-un limbaj nu se aleg la întîmplare, este necesară o mulțime de reguli capabilă să selecționeze cuvintele limbajului, mulțime numită și gramatică formală. Formarea termenilor limbajelor de ordinul întâi se poate considera că are loc după o serie de reguli, date mai jos:

- orice variabilă este un termen;
- orice simbol de constantă individuală este un termen;
- dacă t_1, \dots, t_n sînt termeni iar f este un simbol funcțional n -ar atunci $f(t_1, \dots, t_n)$ este un termen;

- orice aplicare a regulilor de mai sus de un număr finit de ori determină un nou termen.

Formulele atomice ale limbajului se obțin prin aplicarea următoarei reguli: dacă R este un simbol relațional n -ar și t_1, t_2, \dots, t_n sînt termeni, atunci $R(t_1, \dots, t_n)$ este o formulă atomică.

Formarea formulelor limbajului \mathcal{L}_σ se realizează după regulile:

- orice formulă atomică este o formulă;
- dacă $F1, F2$ sînt formule atunci $\neg F1, \neg F2, F1 \wedge F2$ sînt formule;
- dacă x este o variabilă și F este formulă, atunci $\exists xF$ este o formulă;
- orice aplicare a regulilor de mai sus de un număr finit de ori determină o nouă formulă.

O variabilă este liberă într-o formulă dacă:

- x apare într-o formulă atomică F ;
- dacă x este o variabilă liberă în formula F , considerînd și o altă variabilă y din F îndeplinind condiția $y \neq x$, atunci x este o variabilă liberă și pentru formula $\exists yF$;
- dacă x este o variabilă liberă în F , atunci x este de asemenea o variabilă liberă și pentru formulele $\neg F$ și $F \wedge G$ (în care G este orice formulă).

Se numește "expresie bine formată" orice formulă în \mathcal{L}_σ . Se numește *enunț* o formulă care nu conține variabile libere. O variabilă cuantificată universal sau existențial, este o variabilă legată în formula prefixată de cuantificatorul respectiv.

Dacă A este o formulă și x_1, x_2, \dots, x_n sînt variabile ce apar în A se poate scrie relația dintre numele A al formulei și variabilele sale sub forma $A(x_1, x_2, \dots, x_n)$. Se definește substituția σ ca o listă finită de perechi cuprinzînd asociații ale altor variabile, $\sigma = (x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n)$. Dacă $A(x_1, \dots, x_n)$ este o formulă, aplicarea substituției asupra formulei A , notată σA , se realizează prin înlocuirea fiecărei apariții libere a lui x în A prin t_k pentru $k = \overline{1, n}$

$$\sigma A = A(x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n) = A(t_1, \dots, t_n)$$

Se va nota $\text{DOM}(\sigma)$ pentru o substituție σ , domeniul substituției

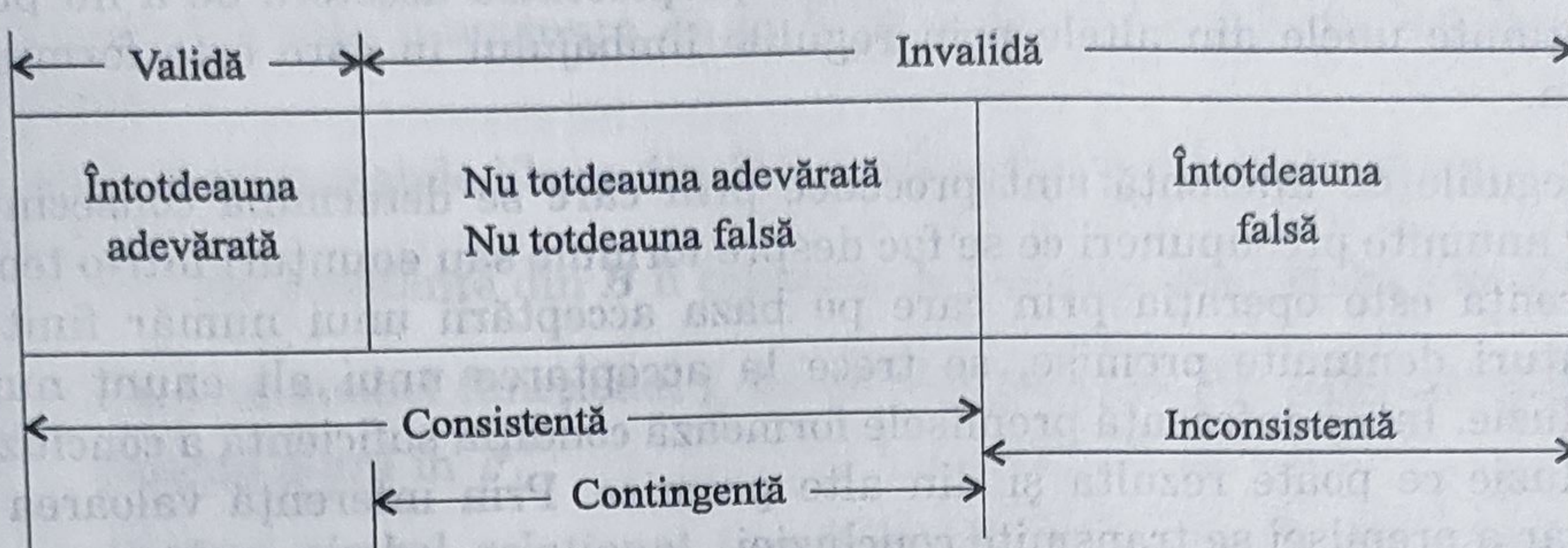
$$\text{DOM}(\sigma) = \{x_i \mid x_i \# \sigma x_i\}$$

și respectiv

$$\text{REP}(\sigma) = \{t_i \mid \sigma x_i = t_i\}$$

mulțimea înlocuitorilor substituției respective.

▼ Fig. 2.1. Relația între valoarea de adevăr și termenii de clasificare a validității unei formule



În semantica oricărui limbaj de ordinul întâi, oricărei formule i se asociază o semnificație și o valoare de adevăr. Mulțimea valorilor de adevăr este specifică limbajului. Pentru logica calculului proporțional și în calculul cu predicate de ordinul întâi, mulțimea este formată din două valori "adevărat" și "fals". Se spune că o expresie este validă dacă valoarea sa de adevăr este totdeauna "adevărat", indiferent de valorile de adevăr ale componentelor sale. O expresie este consistentă dacă valoarea de adevăr nu este întotdeauna "falsă". O expresie este invalidă dacă valoarea sa de adevăr nu este întotdeauna "adevărat". Dacă valoarea de adevăr a unei expresii este întotdeauna "fals" se spune că expresia este inconsistentă.

Așa cum se observă există o categorie de expresii ce nu sînt totdeauna valide și nici totdeauna inconsistente. Aceste expresii care pentru unele combinații ale valorilor de adevăr ale formulelor atomice iau valoarea "adevărat" iar pentru altele iau valoarea "fals", sînt denumite expresii contingente (fig. 2.1).

O mulțime T consistentă de enunțuri într-un limbaj \mathcal{L}_0 este o teorie în acest limbaj. O teorie T_1 este inclusă în altă teorie T_2 ($T_1 \subset T_2$) dacă orice consecință logică în teoria T_1 este o consecință logică și în T_2 . Două teorii T_1 , T_2 sînt echivalente dacă $T_1 \subset T_2$ și $T_2 \subset T_1$. Dacă pentru orice enunț S din limbajul \mathcal{L}_0 atît S cît și $\neg S$ sînt deductibile din T , avem fie $T \vdash S$ fie $T \vdash \neg S$, și se spune că teoria T este completă. Data fiind o structură α de tip σ , o teorie completă T pe această structură este mulțimea tuturor enunțurilor din \mathcal{L}_0 care sînt valide în α . Limbajul formal de ordinul întâi \mathcal{L}_0 în care sînt formulate teoriile despre structurile cu tipul de similaritate σ este denumit limbaj obiect pentru că enunțurile sale sînt obiecte ale acestei teorii. O teorie în care obiectele sînt expresii din \mathcal{L}_0 sau definiții ale obiectelor sale primitive, sau regulile sale de formare pentru formule și expresii bine formate se numește metateorie, iar limbajul asociat metateoriei, metalimbaj peste \mathcal{L}_0 .

Axiomele - sînt formule cu statut de permanentă valabilitate într-o teorie. Mulțimea axiomelor formează ceea ce se numește schema axiomatică.

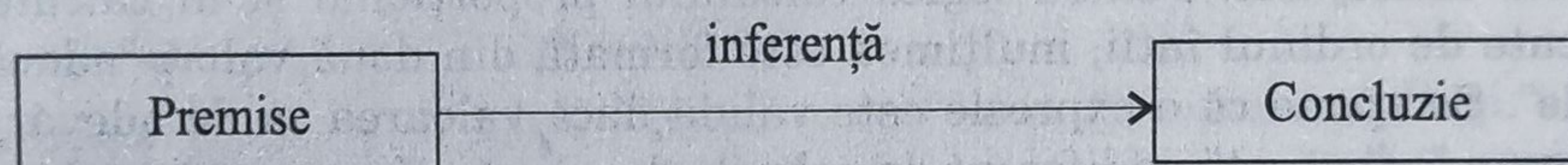
Non-consistența sau *consistența formală* a unei teorii presupune lipsa contradicțiilor adică existența unei formule și a negației sale.

Independența axiomelor - reprezintă proprietatea acestora de a nu putea fi obținute unele din altele prin regulile limbajului în care este formulată teoria.

Regulile de inferență sînt procedee prin care se determină consecințele unor anumite presupuneri ce se fac despre formule sau enunțuri într-o teorie. Inferența este operația prin care pe baza acceptării unui număr finit de enunțuri denumite premise, se trece la acceptarea unui alt enunț numit concluzie. Într-o inferență premisele formează condiția suficientă a concluziei, concluzie ce poate rezulta și din alte premise. Prin inferență valoarea de adevăr a premisei se transmite concluziei.

O teorie pentru care s-au definit axiome și reguli de inferență este o teorie *axiomatică deductivă*. În general o inferență are o structură ca cea din fig. 2.2:

▼ Fig. 2.2. Structura unei inferențe



Un lanț inferențial exprimat în limbaj obiect, ale cărui premise sînt axiome se numește *demonstrație* iar concluzia se numește teoremă. Teoria demonstrației este formată din metodele prin care regulile de inferență sînt aplicate eficient pentru a deduce teoreme din axiome prin "înlănțuire înainte", sau din metodele ce determină validitatea unor teoreme enunțate prin "înlănțuire înapoi". O metodă de bază a inteligenței artificiale este cea de transformare a enunțurilor în enunțuri a căror validitate este demonstrată.

Altă modalitate de abordare a problemelor în vederea soluționării ia în considerație caracteristicile semantice, exprimate de regula în metalimbaj. În acest scop se concepe o construcție formală numită model prin care se pun în corespondență caracteristicile modelului cu cele ale problemei pe care o modelează. Dacă ansamblul concluziilor rezultate din evaluarea modelului este susceptibil de a fi controlat prin determinări experimentale, calcule sau observații, acest ansamblu se constituie în ceea ce se numește teoria modelului respectiv. Obiectele teoriei generale a modelelor sînt structurate. Modelele în inteligența artificială nu necesită realizări fizice, ele fiind pur conceptuale și funcționează pe baza interpretării logice.

Se consideră că limbajele obiect cel mai larg utilizate în inteligența artificială sînt limbajul calculului proporțional și limbajul calculului cu predicate de ordinul întâi. Metalimbajul logicii matematice, în care sînt construite modelele teoretice, cuprinde pe lîngă limbajul obiect și unele simboluri metalingvistice avînd ca scop asertarea validității (\models), asertarea deductibilității (\vdash), valori de adevăr (a-adevărat și f-fals).

Considerînd o formulă B și o structură dată:

$$\alpha = \langle A, R_i^\alpha, F_j^\alpha, e_k^\alpha \rangle \quad i \in I, j \in J, k \in K$$

se spune că φ este o asignare de la B la α dacă următoarele condiții sînt satisfăcute:

- ▶ fiecărei variabile libere din B îi corespunde un element din A ;
- ▶ fiecărei constante din B îi corespunde un element distinct din e_k^α ;
- ▶ fiecărui simbol funcțional f_j de aritate $\psi(s) = n$ din B îi corespunde o funcție n -ara în F_j^α ;
- ▶ fiecărui simbol relațional (predicat) P_i de aritate $\theta(i) = m$ din B îi corespunde un simbol relațional (predicat) m -ar în R_i^α ;
- ▶ fiecare simbol propozițional are o valoare de adevăr, a (adevărat) sau f (fals).

O interpretare a formulei B este o pereche ordonată (α, φ) , în care α este o structură, iar φ este o asigurare de la B la α . Prin interpretarea formulei B , acesteia i se asociază o semnificație în structura α . Unei interpretări (α, φ) a unei formule B i se poate calcula valoarea de adevăr, proces numit și *valuare* a lui B sub asignarea φ în structura α .

2.2. Limbajul calculului propozițional

Calculul propozițional poate fi considerat ca nucleu al oricărui sistem formal. Prima tratare axiomatice a logicii propozițiilor a fost realizată de matematicianul german Gottlob Frege în lucrarea *Begriffsschrift* (1879), însă cu un simbolism greoi ce nu s-a putut impune. Alfred Whitehead și Bertrand Russel în lucrarea *Principia Mathematica* (1910-1913) au o contribuție importantă la desăvîrșirea formulării axiomatice-deductivă a logicii matematice, definind un simbolism mai facil ce este păstrat și astăzi. Totuși utilizarea practică este greoaie, notația nu păstrează semnificația proceselor inferențiale, sistemul axiomatic nu este independent. Dezvoltările ulterioare făcute de David Hilbert, W. Ackermann și Paul Bernays fac calculul propozițional mult mai lucrativ.

Principalele obiective ale introducerii unui sistem formal axiomatic pentru o teorie logică sînt: definirea alfabetului sau a vocabularului, prezentarea regulilor de formare a formulelor sau cuvintelor admise în limbaj, introducerea definițiilor sau a regulilor de prescurtare, enunțarea axiomelor sau a formulelor admise inițial, prezentarea regulilor de inferență sau a regulilor de derivare a formulelor valide, demonstrarea teoremelor. Cercetările ulterioare conduc la verificarea proprietăților metateoretice ale sistemelor axiomatice pentru logica propozițiilor sau ale altor sisteme logice: nonconcordanța, completitudinea, independența.

Vocabularul sau simbolurile primitive ale sistemului axiomatic Hilbert Ackermann sînt:

$$A = \{a, b, \dots, x, \neg, \vee, (,)\}$$

în care:

- ▶ a, b, \dots, x sînt variabile individuale (propoziții) cu două valori de adevăr (adevărat și fals). Se va considera propoziția ca o entitate primitivă ce are o singură proprietate aceea de a lua o singură valoare, și anume valoarea de adevăr;
- ▶ variabile propoziționale formate din negație (\neg), disjuncție (\vee). Negația este o operație monadică, disjuncția o operație binară;
- ▶ semne de punctuație și grupare.

Deci alfabetul conține variabile propoziționale, conective logice și semne de grupare. Prin definiție se pot introduce și ale conective logice. Din considerente de comoditate și lizibilitate în lucrare se vor utiliza și alte conective logice.

Pentru formarea formulelor în limbajul calculului propozițional se utilizează reguli de construcție pornind de la formule elementare ce agregă elementele vocabularului în ansambluri semiotico-sintactice. Se dau mai jos aceste reguli:

- ▶ dacă a și b sînt propoziții, atunci $a; b; \neg a; \neg b; a \vee b; a \wedge b; a \rightarrow b$; sînt formule atomice ale calculului propozițional;
- ▶ orice formulă atomică este o formulă;
- ▶ dacă $F1$ și $F2$ sînt formule atunci $\neg F1; \neg F2; F1 \vee F2; F1 \wedge F2; F1 \rightarrow F2; F1 \equiv F2$ sînt formule;
- ▶ orice formulă inclusă între paranteze este tot o formulă.

Se dau mai jos cîteva dintre axiomele calculului propozițional:

$= A \vee A \rightarrow A$	tautologie
$= A \rightarrow A \vee B$	adițiune
$A \vee B \rightarrow B \vee A$	comutativitate
$(A \rightarrow B) \rightarrow ((C \vee A) \rightarrow (C \vee B))$	principiul însumării

Din combinații sintactice valide de propoziții se pot alcătui trei tipuri de formule:

- ▶ *tautologii* - formule ce au tot timpul valoarea "adevărat" indiferent de valoarea de adevăr a propozițiilor componente;
- ▶ *contradicții* - formule ce au totdeauna valoarea "fals" indiferent de valoarea de adevăr a propozițiilor componente;
- ▶ *formule variabile* - ale căror valori de adevăr sînt dependente de valoarea de adevăr a propozițiilor componente.

Pentru judecăți și aprecieri asupra formulelor scrise în limbajul calculului propozițional se folosesc semne metalingvistice:

- ♦ $\models A$ - indică faptul că A este validă, adică în tabela sa de adevăr se află numai valoarea a , indiferent de valorile de adevăr ale propozițiilor atomice componente;
- ♦ $A \models B$ indică faptul că B este o consecință validă a formulei A și deci B are valoarea "adevărat" peste tot unde A are valoarea "adevărat";
- ♦ $\vdash A$ semnifică faptul că A este deductibilă din axiome sau alte formule prin aplicarea regulii de inferență "modus ponens";
- ♦ $A \vdash B$ - formula B este deductibilă din formula A , adică B se obține ca o secvență finită de formule între care se găsește și A .

Pentru a se obține consecințele asignării unui set de valori de adevăr pentru variabilele propoziționale A, B, C , variabilele ce sînt componente elementare ale unui model exprimat prin formula $(A \vee B) \rightarrow (B \wedge C)$ se determină semnificația fiecărui grup distinct asociat variabilelor A, B, C în modelul teoretic.

Se exemplifică mai jos considerînd că valorile de adevăr pentru A, B, C sînt f, a, f .

▼ Fig. 2.3 Determinarea semnificației

$$\frac{(A \vee B) \rightarrow (B \wedge C)}{(f \vee a) \rightarrow (a \wedge f)} \qquad \frac{a \rightarrow f}{f}$$

Rezultatul obținut este determinat de aplicarea în mod repetat a unei reguli de substituție ce rezultă din teorema:

Fie F o formulă care conține numai propozițiile P_1, \dots, P_n și fie A_1, \dots, A_n formule în calculul propozițional. Fie F' obținut din F prin substituția lui P_1, \dots, P_n simultan cu formulele A_1, \dots, A_n . Dacă $\models F$, rezultă de asemenea $\models F'$. Cu alte cuvinte substituția conservă adevărul.

Prin teorema de mai sus se pot determina formule valide (tautologii) de importanță mare pentru calculul propozițional. Aceste tautologii pot fi considerate drept o schemă axiomatică a unui sistem de deducție naturală bazat pe calculul propozițional. Pentru un sistem de deducție naturală este caracteristic faptul că oferă pentru fiecare conectivă axiome din care se deduc reguli de introducere și reguli de eliminare, ca sistem de reguli eficace în IA:

- ♦ regula de \rightarrow introducere arată că dacă $\Gamma, A \vdash B$, atunci $\Gamma \vdash A \rightarrow B$;
- ♦ regula de \rightarrow eliminare arată că dacă A este adevărat și $A \rightarrow B$ este adevărat, atunci se poate afirma că B este adevărat

$$A, A \rightarrow B \vdash B$$
 (regula este cunoscută și sub denumirea de "modus ponens")
- ♦ regula de \wedge - introducere arată că dacă A este adevărat și B este adevărat, atunci $A \wedge B$ este adevărat, adică $A, B \vdash A \wedge B$;

- ♦ *regula de \wedge - eliminare* arată că dacă $A \wedge B$ este adevărat, se afirma că A este adevărat și B este adevărat, adică:

$$\begin{aligned} A \wedge B &\vdash A \\ A \wedge B &\vdash B \end{aligned}$$

- ♦ *regula de \vee - introducere* arată că dacă A este adevărat, atunci se poate afirma că pentru orice B avem următoarele formule adevărate:

$$A \vdash A \vee B$$

$$A \vdash B \vee A \text{ (conform comutativității);}$$

- ♦ *regula de \vee - eliminare* arată că dacă C se deduce dintr-un lanț inferențial $(\Gamma, A \vee B)$ a cărui ultimă formulă este $A \vee B$, atunci C se poate deduce atât dintr-un lanț inferențial (Γ, A) cât și dintr-un lanț inferențial (Γ, B)

$$(\Gamma, A \vee B \vdash C) \rightarrow (\Gamma, A) \vdash C$$

$$(\Gamma, A \vee B \vdash C) \rightarrow (\Gamma, B) \vdash C$$

- ♦ *regula de \neg - introducere*, arată că dacă dintr-un lanț inferențial (Γ, A) se deduce atât B cât și negația lui, respectiv $\neg B$, atunci se poate deduce și negația lui A , deci din $(\Gamma, A) \vdash B$ și $(\Gamma, A) \vdash \neg B$ atunci $\Gamma \vdash \neg A$ (inferența cunoscută sub numele de reducere la absurd);

- ♦ *regula de \neg - eliminare*, arată că dacă se neagă negația lui A atunci se afirmă A . Deci, $\neg \neg A \vdash A$ (principiul dublei negații);

- ♦ *regula de \equiv - introducere* arată că dacă este adevărat ca $A \rightarrow B$ și este adevărat că $B \rightarrow A$ atunci A și B sînt echivalente

$$(A \rightarrow B, B \rightarrow A) \vdash (A \equiv B)$$

- ♦ *regula de \equiv - eliminare*, arată că dacă A și B sînt echivalente, sînt simultan adevărate și implicațiile $A \rightarrow B$ și $B \rightarrow A$

$$A \equiv B \vdash (A \rightarrow B)$$

$$A \equiv B \vdash (B \rightarrow A)$$

Deducerea valorii de adevăr a oricărei formule exprimată în limbajul calculului propozițional se poate realiza prin aplicarea de un număr finit de ori a regulilor de inferență. Se spune că o formulă B este o *deducție formală* din formulele A_1, A_2, \dots, A_n ($A_1, A_2, \dots, A_n \vdash B$) dacă B este fie una din formulele A_1, A_2, \dots, A_n sau una din axiomele sistemului formal, sau provine din două formule A_1, \dots, A_n prin aplicarea regulii de inferență *modus ponens*.

Se spune că B este o *consecință validă* a lui A_1, A_2, \dots, A_n dacă prin construirea model teoretică a tabelii de adevăr pentru $A_1, \dots, A_n \models B$ în funcție de valorile de adevăr P_1, \dots, P_m se obține valoarea "a" pentru B în toate cazurile unde A_1, \dots, A_m sînt simultan adevărate.

Teoremă. $A_1, A_2, \dots, A_m \models B$ dacă și numai dacă $A_1, A_2, \dots, A_{m-1} \models A_m \rightarrow B$. Se obține astfel următorul corolar: $A_1, A_2, \dots, A_m \models B$ dacă și numai dacă $\models A_1 \rightarrow (\dots (A_{m-1} \rightarrow (A_m \rightarrow B)) \dots)$.

Demonstrația teoremei se face pornind de la analiza tabelelor de adevăr pentru A_1, \dots, A_m, B și $A \rightarrow B$, în care se substituie A_m lui A apoi se selectează numai acele linii în care A_1, \dots, A_{m-1} sînt simultan adevărate.

Teorema deducției. Dacă $A_1, \dots, A_{m-1}, A_m \vdash B$ atunci $A_1, \dots, A_{m-1} \vdash A_m \rightarrow B$.

Se obține din această teoremă corolarul. Dacă $A_1, \dots, A_{m-1}, A_m \vdash B$ atunci $\vdash A_1 \rightarrow (\dots (A_{m-1} \rightarrow (A_m \rightarrow B) \dots))$ ce sînt rezultate de mare importanță pentru teoria deducției.

2.3. Limbajul calculului cu predicate de ordinul întâi

Termenul de predicat a fost utilizat pentru prima dată de Aristotel în teoria sa silogistică. În viziunea sa predicatul era un termen al unei judecăți și anume cel care descrie ce se afirmă despre subiect. În sens aristotelic o judecată are forma "Socrate e muritor", în care "Socrate" și "muritor" sînt predicate. În logica matematică termenul de predicat este folosit în accepțiunea de formulă deschisă sau funcție logică în forma muritor (x), tata (x), fiu(x, y), etc. Formulele de acest tip pot deveni enunțuri sau propoziții dacă variabilele iau ca valori constante individuale sau sînt cuantificate universal ori existențial. Logica predicatelor așa cum i-a fost atribuită lui G. Frege a avut scopul de creare a unui limbaj capabil să descrie enunțurile matematice și să garanteze corectitudinea raționamentelor. Limbajul logicii predicatelor dispune de instrumente pentru a defini tipuri de obiecte matematice cum sînt: numere, variabile numerice, funcții și operații matematice, relații, propoziții și teoreme matematice. În logica predicatelor s-au standardizat schemele de inferență ce asigură trecerea de la enunțuri adevărate sau premise la alte enunțuri adevărate numite și consecințe sau concluzii.

În mod normal oamenii nu gîndesc în limbajul teoriilor logice, ci în limbile naturale, în raport cu scopurile pe care le urmăresc. Este greu în a garanta corectitudinea raționamentelor efectuate, fiind dependent de opinia celui ce evaluează raționamentul. A testa validitatea unui astfel de raționament este o chestiune dificilă, chiar dacă logica matematică modernă dispune de mijloace adecvate, întrucît este mai întîi necesară transpunerea limbajului natural într-un formalism logico-formal.

Fiecare teorie logică are particularități în ceea ce privește semnele primitive (alfabetul), operațiile logice admise, numărul și natura valorilor de adevăr, principiile și metodele prin care se leagă formulele rezultate cu elementele lumii reale. În acest paragraf se urmărește prezentarea principalelor componente ale limbajului calculului cu predicate, semantica logicii predicatelor, teoria inferențelor logice.

În principal se poate afirma ca limbajul calculului cu predicate de ordinul întâi reprezintă o extensie a calculului propozițional prin luarea în considerație a componentelor propozițiilor, pentru a interpreta modul în care acestea

participă la determinarea valorilor de adevăr ale formulelor. Extensia se referă la simbolurile primitive ale limbajului. Propoziția este considerată a avea o structură cu două componente:

- ▶ *predicatul* - ce specifică semnificația funcțională a propoziției;
- ▶ *argumentul predicatului* - specifică semnificația suportului funcțional al propoziției.

Orice predicat $P(x)$ joacă rolul de funcție propozițională deoarece pentru orice valoare a argumentului independent x , acesta ia ca valoare o propoziție, cu valoarea adevărat sau fals. Numărul de obiecte individuale ce compun argumentul reprezintă aritatea predicatului. Astfel pentru:

$n = 0$ - predicatul este o propoziție;

$n = 1$ - predicatul specifică o proprietate a argumentului;

$n = 2$ - predicatul specifică o proprietate binară;

$n = 3$ - relație n -ară.

Orice predicat este de ordinul întâi dacă variabilele sale nu sînt predicate (se poate spune despre astfel de variabile că pot fi considerate predicate de ordinul zero). Un predicat care are cel puțin una din variabilele sale individuale predicate de ordinul întâi se spune că este un predicat de ordinul doi. Se poate spune în general că un predicat este de ordin n dacă cel puțin una din variabilele sale este un predicat de ordinul $n - 1$. Pentru calculul cu predicate de ordinul întâi formulele se formează după regulile limbajului calculului propozițional, în care variabilele propoziționale sînt înlocuite cu predicate. În relația folosită pentru predicate, variabilele au semnificație nu prin simbolurile folosite, ci prin poziția și rolul pe care îl joacă în definirea predicatului respectiv. Variabilele folosite în aceste notații sînt denumite variabile formale, iar notația definitorie pentru predicate este denumită *expresie predicativă primară sau ion*.

Se spune că $P(x, y)$ și $P(u, v)$ sînt diferite atașări de variabile formale pentru aceeași expresie predicativă primară $P(_1, _2)$, iar $P(a, b)$ este o instanță a acestei expresii predicative în care a și b au fost asignate primului și respectiv celui de-al doilea argument al predicatului. Pentru aprecierea valorii de adevăr în raport cu domeniul în care iau valori variabilele individuale ale predicatelor componente, se introduc o serie de operatori unari denumiți și cuantificatori ce acționează asupra variabilelor.

- ▶ cuantificatorul universal indică faptul că formula prefixată este validă pentru toate valorile posibile în domeniul variabilei x .
- ▶ cuantificatorul existențial indică faptul că formula prefixată este validă pentru cel puțin o valoare din domeniul variabilei x .

Orice formulă bine formată prefixată cu un cuantificator (\exists sau \forall) aplicat unei variabile individuale care apare în formulă, este o formulă bine formulată. O formulă F prefixată de cuantificatori universali sau existențiali, cum sînt $\exists xF$ și $\forall xF$ este denumită *scop* al cuantificatorului respectiv. Rezultă deci, că în limbajul calculului cu predicate de ordinul întâi orice formulă bine formată, prefixată cu un cuantificator (\exists sau \forall) aplicat unei variabile individuale care apare în formulă, este o formulă bine formulată.

Teoria demonstrației la calculul cu predicate de ordinul întâi folosește pe lângă axiomele și regulile de inferență ale limbajului calculului propozițional și următoarele axiome:

$$\models \forall x A(x) \rightarrow A(r)$$

$$\models A(r) \rightarrow \exists x A(x) \text{ în care:}$$

x este orice variabilă

$A(x)$ este o formulă

r - orice variabilă nu neapărat distinctă de x

$A(r)$ - rezultatul substituției libere cu r a oricărei apariții libere a variabilei x în $A(x)$.

Reguli de inferență pentru calculul cu predicate de ordinul întâi:

$$(\forall) \text{ regula: dacă } \models C \rightarrow A(x) \text{ atunci } \models C \rightarrow \forall x A(x)$$

$$(\exists) \text{ regula: dacă } \models A(x) \rightarrow C \text{ atunci } \models \exists x A(x) \rightarrow C$$

în care: x - orice variabilă; $A(x)$ o formulă; C este o formulă ce nu conține variabila liberă x .

În general se spune că substituția lui x cu r în $A(x)$ este liberă, dacă pentru fiecare apariție liberă a lui x în $A(x)$ rezultă o apariție liberă corespunzătoare a lui r în $A(r)$.

Un predicat într-o formulă depinde de atâtea variabile câte litere apar libere în el. Ca exemplu în formula:

$$\forall y \exists w P(y, w, z) \rightarrow \forall x \exists w P(x, w, z)$$

predicatul 3-*ar* P depinde de o singură variabilă în ambele sale apariții deoarece la prima apariție variabilele y și w sînt legate, singura variabilă liberă fiind z iar la a doua apariție variabilele x și w sînt legate singura variabilă liberă fiind tot z .

Orice formulă ce nu conține apariții libere ale variabilelor sale, se numește formulă inclusă. Cînd includerea unei formule este dată numai de unele din variabilele sale este recunoscută ca închiderea la variabilele respective. Dacă unei formule i se adaugă cuantificatorul universal pentru fiecare variabilă liberă, devine închiderea universală a formulei considerate \mathcal{F} . O formulă \mathcal{F} este o *consecință validă* a formulelor A_1, \dots, A_m relativ la variabilele libere x_1, x_2, \dots, x_r și se notează:

$$A_1, \dots, A_m \models x_1, \dots, x_r \mathcal{F}$$

dacă se consideră constante toate celelalte variabile, cu excepția celor specificate. Similar, o formulă este *deductibilă* din A_1, \dots, A_m relativ la variabilele x_1, \dots, x_r și se notează

$$A_1, \dots, A_m \vdash x_1, \dots, x_r \mathcal{F}$$

dacă se consideră deductibilitatea din formulele A_1, A_2, \dots, A_m cu menținerea constantă a tuturor celorlalte variabile.

Calculul cu predicate de ordinul întâi se bazează pe rezultatele unor teoreme importante cum sînt:

- ▶ teorema de substituire pentru expresii predicative primare;
- ▶ teorema de substituire pentru variabile individuale;
- ▶ teoreme de \forall - introducere, \forall - eliminare, \exists - introducere, \exists - eliminare.

2.4. Teoria inferențelor logice

A decide asupra unei formule înseamnă a determina printr-o metodă adecvată dacă formula este validă sau infirmată. A. Chaurk (1936) demonstrează faptul că logica predicatelor nu este o teorie complet decidabilă, deci nu există o metodă universală prin care într-un număr finit de pași să decidă dacă o formulă bine formată este validă sau infirmată. Se poate afirma însă, că există diferite clase de formule cum sînt forma normală prenex, forma normală Skolem ce pot fi descrise într-un număr finit de pași. Metode cum sînt metodele tablourilor semantice sau metoda arborilor de decizie nu vor fi prezentate aici, atenție deosebită fiind acordată metodei rezoluției.

Modul de desfășurare a raționamentelor pentru obținerea de concluzii valabile din premisele enunțate în probleme, este obiectivul oricărui sistem inteligent de rezolvare a problemelor. Raționamentele efectuate asupra formulelor exprimate în limbajul logicii matematice, cuprind o serie de raționamente elementare specifice numite reguli de inferență. Din analiza limbajului calculului cu predicate de ordinul întâi și a enunțurilor (axiome și reguli) exprimate în acest limbaj se definește o teorie a calculului cu predicate de ordinul întâi notată T . Se observă că enunțurile sînt de 3 feluri:

- ▶ enunțuri referitoare la propoziție, care alcătuiesc o teorie a propozițiilor T_1 ;
- ▶ enunțuri referitoare la predicate, care alcătuiesc o teorie a predicatelor T_2 ;
- ▶ reguli de inferență ce alcătuiesc o teorie a demonstrației T_3 .

Există evident relații $T_1 \subset T$, $T_2 \subset T$, $T_3 \subset T$ și $T = T_1 \cup T_2 \cup T_3$.

Teoria inferențelor logice este formată din abstracția tuturor raționamentelor elementare a căror valabilitate nu depinde de obiectele implicate în raționament ci doar de structura premiselor. Gerhard Gentpen face prima abstractizare științifică ce permite o definire sistematică a regulilor de inferență, formalizînd schemele de derivare pe baza cărora se pot construi mecanismele inferențiale în sistemele cognitive și rezolutive. Dacă s-ar considera că un sistem de reguli de inferență este consistent cu sistemul axiomatic al unei teorii și deci nu ar mai fi necesar, ar rezulta mari dificultăți în utilizarea instrumentelor logicii matematice, deoarece derivarea formulelor

într-un sistem axiomatic ce folosește ca singură inferență substituția determină calcule laborioase. O astfel de abordare va atrage după sine imposibilitatea folosirii lanțului de transformări operate drept justificare a unui raționament. Semnificația operațională este dată de faptul că au un conținut semantic. Este motivul pentru care în IA sistemele axiomatic și inferențial trebuie să coexiste. Oricum, principiul rezoluției este o regulă de inferență mai generală decât *modus ponens*, *modus tollens*, principiul raționamentului ipotetic sau legea tranzitivității. Aplicarea metodei necesită mai întâi aducerea la o formă standard numită formă clauzală.

Ca urmare cea mai obișnuită modalitate de reprezentare a regulilor de inferență folosește forma clauzală a expresiilor în limbaje de ordinul întâi. Iată câteva tipuri de clauze:

(i) Dacă $A_1 \wedge A_2 \wedge \dots \wedge A_m$, atunci B

(ii) B , dacă $A_1 \wedge A_2 \wedge \dots \wedge A_m$

Ambele forme descriu aceeași clauză avînd o consecință unică și un antecedent complex.

(iii) Dacă $A_1 \wedge A_2 \wedge \dots \wedge A_m$, atunci $B_1 \vee B_2 \vee \dots \vee B_n$

(iv) $B_1 \vee B_2 \vee \dots \vee B_n$, dacă $A_1 \wedge A_2 \wedge \dots \wedge A_m$

(v) $A_1, A_2, \dots, A_m \rightarrow B_1, B_2, \dots, B_n$

(vi) $B_1, B_2, \dots, B_n \leftarrow A_1, A_2, \dots, A_m$

În (iii) - (vi) au fost prezentate în dublă scriere clauze cu consecințe alternative. În (v) și (vi) s-au omis conectivele de conjuncție la premisă și disjuncție la concluzie.

Forma generală a unei clauze complexe este dată în (vii)

(vii) Pentru orice x_1, \dots, x_k $B_1, B_2, \dots, B_n \leftarrow A_1, A_2, \dots, A_m$ unde x_1, \dots, x_k sînt variabile individuale, B_1, B_2, \dots, B_n sînt concluzii alternative, A_1, A_2, \dots, A_m sînt condiții valabile simultan.

Se va defini un literal ca o formulă atomică sau negația ei. Se numesc literalii pozitivi formulele atomice și literalii negativi negația acestora. Cu aceste precizări o clauză în logica predicatelor, este un set de literalii conectați prin disjuncție. De regulă în scrierea clauzei se omite semnul disjuncției aceasta apărînd ca mulțimi de literalii. Tipologia clauzelor este și ea variată. Pot exista clauze alcătuite dintr-un singur termen numite și clauze unitare, clauze cu n literalii, clauze negative, clauze Horn. Se numesc clauze Horn acele clauze ce conțin cel mult un literal pozitiv. Se poate ușor observa că o clauză Horn provine dintr-o implicație de forma

$$A_1, A_2, \dots, A_m \vdash B$$

unde A_1, \dots, A_m sînt condițiile și B consecința.

Cu aceste precizări este important a arăta că orice formulă bine formată și închisă în logica predicatelor poate fi adusă la forma clauzală. Pentru a se aduce o formulă din logica predicatelor de ordinul întâi la forma clauzală este necesară parcurgerea următoarelor etape:

- (i) eliminarea implicațiilor și echivalentelor;
- (ii) coborîrea implicațiilor pe semnele predicative;
- (iii) standardizarea variabilelor;

- (iv) eliminarea cuantificatorilor existențiali;
- (v) eliminarea cuantificatorilor universali;
- (vi) reducerea expresiei la conjuncții de clauze disjunctive;
- (vii) eliminarea operatorilor propoziționali;
- (viii) redenumirea variabilelor.

Eliminarea implicațiilor și echivalentelor are loc pe baza relațiilor de mai jos

$$A \supset B \equiv \neg A \vee B$$

$$A \subset B \equiv A \vee \neg B$$

$$\equiv (A \equiv B) \equiv (\neg A \vee B) \wedge (A \vee \neg B).$$

Al doilea pas se bazează pe legile negației

$$\neg \neg A \equiv A$$

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$\neg(\forall x) A(x) \equiv \exists x \neg A(x)$$

$$\neg(\exists x) A(x) \equiv \forall x \neg A(x)$$

Pasul de standardizare a variabilelor cere ca o variabilă într-o formulă să fie cuantificată într-un singur fel. În acest context nu se permite ca o variabilă într-o formulă să fie cuantificată atât existențial cât și universal ca în $\exists y Py \wedge \forall y Qy$. Formula se va rescrie $\exists y Py \wedge \forall z Qz$, cu z o variabilă individuală oarecare, diferită de y .

Pasul privind eliminarea cuantificatorului existențial se sprijină pe teorema Skolem și pot fi utilizate următoarele reguli:

- 1) Dacă un cuantificator existențial nu este precedat în formula dată de nici un cuantificator universal se va elimina variabila de sub cuantificatorul existențial și în formulă va fi înlocuită cu o constantă ce nu mai apare. Spre exemplificare se consideră formula $\exists x (P(x) \wedge R(x))$ în care variabila prefixată este x și nu este precedată de cuantificatori universali. Conform strategiei prefixul $\exists x$ va fi eliminat și o constantă c , nouă, va fi dispusă în locul aparițiilor lui x , obținînd $P(c) \wedge R(c)$.
- 2) Dacă cuantificatorul existențial este precedat în prefix de cuantificatori universali se va elimina cuantificatorul existențial din prefix și orice apariție în formulă a variabilei prefixate va fi substituită printr-un semn funcțional ce depinde de toate variabilele de sub cuantificatorii universali ce preced cuantificatorul existențial în discuție. Dacă se consideră formula $\forall x \forall y \exists z (P(x, y, z) \wedge R(x, z))$ se observă că $\exists z$ este precedat de cuantificatorii universali $\forall x$ și $\forall y$. Urmînd regula de modificare se va elimina din prefix z iar z se va înlocui prin $f(x, y)$, obținînd $\forall x \forall y (P(x, y, f(x, y)) \wedge R(x, f(x, y)))$.

Cuantificatorii universali rămași în formule vor fi eliminați conform convențiilor de rescriere și interpretare semantică, iar celelalte etape sînt destul de simple.

Se obține deci pentru forma generală a regulilor de inferență expresia:

$$\frac{A_1 \dots A_m}{B_1 \dots B_n} \quad (m > 1, n > 1)$$

în care A_1, A_2, \dots, A_m sînt clauzele ce alcătuiesc premisa regulii, iar B_1, B_2, \dots, B_n sînt clauzele ce alcătuiesc concluzia. Pornind de la această regulă generală o regulă de inferență este o clauză de clauze. O clauză se interpretează ca o concluzie alternativă, condiționată de valabilitatea simultană a formulelor ce alcătuiesc premisa.

Cazuri extreme:

- ▶ $m = 0 \vdash B_1, B_2, \dots, B_r$ adică în mod necondiționat este adevărat B_1, B_2, \dots, B_r adică avem o aserțiune;
- ▶ $n = 0 \ A_1, A_2, \dots, A_m \vdash$ indicînd faptul că din $A_1 A_2 \dots A_m$ se deduce falsul întrucît nu se deduce nimic; deci se obține o negare a premisei;
- ▶ $n = m = 0$ se obține clauza vidă \emptyset adică o propoziție falsă.

Pe baza celor de mai sus se pot clasifica regulile de inferență în:

- a) *reguli de inferență structurală* - care se referă la structura clauzelor, indiferent de natura conectivelor ce apar în formulele componente;
- b) *reguli de inferență logică* - reguli specifice pentru introducerea sau eliminarea conectivelor logice.

Se numește regulă de demonstrare o succesiune de reguli de inferență ce satisfac condițiile:

- ▶ formulele ce apar în concluzia unei reguli figurează în cel mult o premisă a altei reguli;
- ▶ fiecare formulă exceptînd eventual o formulă finală apare în premisa cel puțin a unei reguli;
- ▶ regulile în ansamblu nu conțin circularități, adică formulele nu generează cicluri.

Acele formule care nu apar în concluzia unei reguli de inferență se numesc formule inițiale.

Un *fir demonstrativ* este format dintr-o succesiune de formule componente ale regulii de demonstrare ce satisfac condițiile:

- ▶ prima formulă a succesiunii este o formulă inițială;
- ▶ ultima formulă a succesiunii este o formulă finală;
- ▶ fiecare premisă (cu excepția ultimei) este o premisă pentru acea regulă de inferență a cărei concluzie este chiar premisa regulii următoare.

2.4.1. Reguli de inferență structurală

Dacă A, B, C, D sînt formule oarecare și $\Gamma, \Delta, \theta, \psi$, șiruri de formule oarecare separate prin virgulă, conectiva implicită între formulele individuale este cea care corespunde poziției pe care acestea le ocupă în clauză, (conjunția pentru antecedent și disjunția pentru succedent). Se pot stabili următoarele reguli de inferență structurală:

Regula de rafinare

$$\triangleright \text{ la antecedent } \frac{\Gamma \vdash \psi}{A, \Gamma \vdash \psi}$$

$$\triangleright \text{ la succedent } \frac{\Gamma \vdash \psi}{\Gamma \vdash \psi, A}$$

Regula de simplificare

$$\triangleright \text{ la antecedent } \frac{A, A, \Gamma \vdash \psi}{A, \Gamma \vdash \psi}$$

$$\triangleright \text{ la succedent } \frac{\Gamma \vdash \psi, A, A}{\Gamma \vdash \psi, A}$$

Regula de intervertire

$$\triangleright \text{ la antecedent } \frac{\Gamma, B, C, \Delta \rightarrow \psi}{\Gamma, C, B, \Delta \rightarrow \psi}$$

$$\triangleright \text{ la succedent } \frac{\Gamma \vdash \psi, B, C, \Delta}{\Gamma \vdash \psi, C, B, \Delta}$$

Regula de eliminare a secționării

$$\triangleright D, \Delta \vdash \wedge$$

$$\triangleright \frac{\Gamma \vdash \psi, D}{\Gamma, \Delta \vdash \wedge, \psi}$$

2.4.2. Reguli de inferență logică

Varietatea acestor reguli este dată de tipul conectivelor logice, de modul în care conectiva logică apare în formule, de poziția din clauza afectată prin inferență și de tipul formulelor componente ale regulii. În funcție de conectivele logice există reguli de inferență pentru conjuncție \wedge , disjuncție \vee , implicație, echivalență, negație, pentru \exists , respectiv \forall .

Modul în care conectiva apare în inferență face separarea între regulile ce introduc conective și regulile de eliminare a conectivelor. Funcție de poziția din clauza afectată avem regulă ce operează la antecedent respectiv la succedent.

După tipul formulei avem:

- ▶ reguli care se aplică la componentele unei formule oarecare;
- ▶ reguli care se aplică în cazul în care componentele sînt reprezentate în forma clauzală.

Dintre aceste reguli se pot remarca:

- ▶ regulile de introducere a conjuncției;
- ▶ regulile de eliminare a conjuncției;
- ▶ regulile de introducere a disjuncției;
- ▶ regulile de eliminare a disjuncției;
- ▶ regulile de introducere a implicației;
- ▶ regulile de eliminare a implicației;
- ▶ regulile de introducere a echivalenței;
- ▶ regulile de eliminare a echivalenței;
- ▶ regulile de introducere a negației;
- ▶ regulile de eliminare a negației;
- ▶ regulile de introducere a cuantificatorilor;
- ▶ regulile de eliminare a cuantificatorilor.

2.5. Tehnici fuzzy

2.5.1. Teoria seturilor fuzzy

Pentru facilitarea introducerii în teoria seturilor fuzzy se pornește de la o scurtă discuție a setului teoretic al funcțiilor de apartenență. În acest scop, se pornește de la existența setului universal X la care alte mulțimi sînt subseturi, ca de exemplu pentru setul $A (A \subset X)$, se spune că $a \in A$ implică $a \in X$. Date fiind postulatele setului universal, orice set A poate fi reprezentat prin utilizarea funcției de apartenență, numită și altfel funcție de adevăr, funcție de evaluare sau funcție caracteristică. Funcția de apartenență $u_A(x)$ a setului A este definită relativ la setul universal X prin:

$$u_A(x) = \begin{cases} 1 & \text{dacă } x \in A \\ 0 & \text{dacă } x \notin A \end{cases}$$

În consecință, orice subset A al lui X poate fi reprezentat în forma $A = \{x \mid u_A(x)\}$.

Specific, fiecare x din X este asociat cu funcția sa de evaluare. Dacă A este finită, poate fi reprezentată ca o listă a tuturor elementelor lui X , fiecare număr urmat de un slash după care, apare fie valoarea 1, fie valoarea 0 după cum elementul aparține lui A sau nu aparține lui A .

Exemplu: Considerînd setul universal $X = \{1, 2, 3, 4\}$, dacă $A = \{2, 4\}$ utilizarea funcției de apartenență va da $A = \{1/0, 2/1, 3/0, 4/1\}$.

Astfel spus într-o reprezentare simbolică fiecare simbol din X are atașat un "flag", dacă indica 1 acesta aparține setului A , iar dacă indică 0 nu aparține setului A .

Este necesară discutarea setului teoretic al operațiilor referitoare la caracterizarea funcției de apartenență. Considerînd două serturi A și B reuniunea $(A \cup B)$ conține toate elementele ce aparțin fie lui A fie lui B . Utilizarea reprezentării standard va duce la

$$A \cup B = \{x \mid x \in A \text{ sau } x \in B\}$$

pe cînd prin funcția de apartenență

$$A \cup B = \{x \mid u_{A \cup B}(x) = \max(u_A(x), u_B(x))\}.$$

Cu X și A din exemplul de mai sus și cu

$$B = \{1/1, 2/1, 3/0, 4/0\} = \{1, 2\}$$

se obține:

$$A \cup B = \{1/1, 2/1, 3/0, 4/1\} = \{1, 2, 4\}.$$

Intersecția dintre seturile A și B , $A \cap B$ este reprezentată de setul elementelor ce aparțin atît lui A cît și lui B , adică:

$$A \cap B = \{x \mid u_{A \cap B}(x), \text{ unde } u_{A \cap B}(x) = \min(u_A(x), u_B(x))\}.$$

Diferența între A și B va fi reprezentată ca

$$A - B = \{x \mid u_{A-B}(x), \text{ unde } u_{A-B}(x) = \min(u_A(x), 1 - u_B(x))\}.$$

În acest caz atunci cînd A este egal cu X , B reprezintă complementarea lui X , notată și

$$B^c = X - A = \{x \mid u_B(x), \text{ unde } u_B^c(x) = 1 - u_B(x)\}$$

deci

$$\min(u_X(x), 1 - u_B(x)) = \min(1, 1 - u_B(x)) = 1 - u_B(x)$$

Întrucît operațiile de intersecție și reuniune sînt comutative și asociative, operația poate fi generalizată pentru orice subset al claselor arbitrare și formulele de Morgan sînt valabile.

Prin setul teoretic, se arată dacă un element aparține sau nu unui set. Ca exemplu, dacă setul universal X este reprezentat de mulțimea tuturor poligoanelor, setul triunghiurilor este definit prin

$A = \{x \mid x \text{ are trei laturi}\}$ este un subset al lui X .

Oricum pentru un particular $x \in X$ sînt posibile două situații exclusive: (1) $x \in A$ în care caz $u_A(x) = 1$, sau (2) $x \notin A$, caz în care $u_A(x) = 0$. Însă excluderea mutuală nu poate fi strict aplicabilă în orice tip de raționare. Dacă se consideră setul animalelor X cu toate speciile sale nu pot fi caracterizate exhaustiv toate speciile pentru a putea spune că un anumit animal face parte din categoria A sau B . Procedînd intuitiv, pentru ca A să fie constituit ca un set propriu este necesar să se formuleze întrebări asupra oricărui element x din X . În general, dacă setul A este definit prin formularea cu predicate

$$A = \{x \mid x \text{ este } \underline{\hspace{2cm}}\}$$

atunci aceasta incumbă faptul că predicatul nu este precis definit. Cu alte cuvinte se atribuie un conținut extrasemantic pentru predicat ținând cont de stricta interpretare relațională. Modelul nedeterminat al apartenenței la clasa introdus de Zadeh, folosește noțiunea de subset fuzzy al setului universal X . Pornind de la acesta subsetul A este definit ca $A = \{x \mid u_A(x)\}$ unde funcția de apartenență u_A poate lua valori în intervalul $[0, 1]$, așa că pentru orice $x \in X$, se obține

$$0 \leq u_A(x) \leq 1.$$

Față de situația clasică apartenența la un subset nu este reprezentată prin valorile precise 0 sau 1. Dacă X reprezintă setul tuturor speciilor, orice element x din X are valoarea $u_B(x)$ indicînd apartenența la subsetul B . Din acest punct de vedere x este mai sigur în specia B decît y dacă $u_B(x) > u_B(y)$. Funcția de apartenență determină o caracterizare cantitativă a calității.

Exemplu: Se consideră setul X al roboților aparținînd diferitelor corporații. Acești roboți formează setul $X = \{a, b, c\}$. Se dorește caracterizarea setului A al roboților inteligenți. Într-o manieră tranșantă se poate spune că robotul a aparține clasei A dacă este considerat inteligent sau nu aparține clasei dacă este complet neinteligent. Utilizarea seturilor fuzzy permite gradarea inteligenței pe baza funcției de apartenență. Cel mai inteligent robot este a , cu cea mai mare valoare a funcției $u_A(a)$. Intuitiv membrii cu valoare apropiată de $1/2$ sînt cei mai dificil de apreciat, adică certitudinea apartenenței sau neapartenței la clasă este cel mai greu de decis. Dacă mulțimea A ce corespunde caracterizării "este inteligent" este:

$$A = \{a/0.2, b/0.9, c/0.4\}$$

se poate afirma că robotul b este cel mai inteligent.

Dacă $u_A(x) = 1$ atunci se poate scrie ca $x \in A$ respectiv dacă $u_A(x) = 0$ $x \notin A$. Dacă însă $0 < u_A(x) < 1$, se scrie $x \in A$. Pornind de la teoria de bază se poate spune pentru un set A că este un subset al lui B (adică $A \subset B$) dacă pentru orice x din X este valabilă inegalitatea $u_A(x) \leq u_B(x)$.

Despre două seturi fuzzy se spune că sînt egale dacă $u_A(x) = u_B(x)$, și se notează $A = B$. Pentru operațiile cu seturile fuzzy între care sînt luate în considerație intersecția, reuniunea, diferența este necesară definirea modului de calcul al funcției de apartenență. Astfel:

$$u_{A \cup B} = \max(u_A, u_B)$$

$$u_{A \cap B} = \min(u_A, u_B)$$

$$u_{A-B} = \min(u_A, 1 - u_B)$$

Față de logica clasică în care două valori de adevăr sînt luate în considerație, în teoria fuzzy este necesară operarea într-o logică cu mai multe valori. Astfel se cunoaște logica cu trei valori Kleene, Lukasiewicz, Bochvar, logică ce poate fi extinsă asupra predicatelor fuzzy (Multi-Valued-Logic; MVL). Se definește un cadru cu mai multe valori L , ca un set nevid M împreună cu setul relațiilor fuzzy pe M (unde la n valori fuzzy pe M se obține o funcție de la M^n la $[0, 1]$). Conectivile logice MVL se vor calcula în următorul mod:

- (i) $\neg r = 1 - r$
- (ii) $r_1 \wedge r_2 = \text{Min}(r_1, r_2)$
- (iii) $r_1 \vee r_2 = \text{Max}(r_1, r_2)$
- (iv) $r_1 \rightarrow r_2 = \text{Min}(1, 1 - r_1 + r_2)$
- (v) $\bigwedge_{i \in I} r_i = \text{Min}_{i \in I}(r_i)$
- (v) $\bigvee_{i \in I} r_i = \text{Max}_{i \in I}(r_i)$.

Ultimele două introduc specificitatea cuantificatorilor universal și respectiv existențial. Cu aceste precizări semantica este similară cu cea din cazul pentru logica cu trei valori. Fie A și B formule în L la care se poate asocia un număr real în intervalul $[0, 1]$ după cum urmează:

$$[A \wedge B] = [A] \wedge [B]$$

$$[\neg A] = \neg([A])$$

$$[A \vee B] = [A] \vee [B]$$

$$[A \rightarrow B] = [A] \rightarrow [B]$$

$$[\forall x A] = \bigwedge_{m \in M} [A(m)]$$

$$[\exists x A] = \bigvee_{n \in M} [A(n)]$$

unde prin $[A]$ s-a notat pentru subsetul fuzzy din intervalul $[0, 1]$, funcția sa de apartenență.

2.5.2. Logica fuzzy

În logica fuzzy (FL) setul valorilor de adevăr în intervalul $[0, 1]$ este înlocuit cu subsetul fuzzy al acelui set. Zadeh, autorul logicii fuzzy nu permite totuși toate subseturile fuzzy. Valorile de adevăr sînt presupuse a fi numărabile. Un astfel de set are forma

VA = (adevărat, fals, neadevărat, foarte adevărat, nu foarte adevărat, mai mult sau mai puțin adevărat, destul de adevărat,...).

Fiecare element al acestui set reprezintă un subset pe intervalul $[0, 1]$. Dacă U_{max} este membru al subsetului fuzzy adevărat se pot genera funcțiile de apartenență pornind de la acesta. Atunci funcțiile de apartenență pentru ceilalți membri din VA se obțin

$$U_{\text{fals}}(v) = U_{\text{adevărat}}(1 - v)$$

$$U_{\text{neadevărat}}(v) = 1 - U_{\text{adevărat}}(v)$$

$$U_{\text{foarte adevărat}}(v) = (U_{\text{adevărat}}(v))^2 \text{ etc.}$$

În acest mod înțelesul valorilor lingvistice este crucial dependentă de înțelesul asociat pentru adevărat. Zadeh motivează această alegere prin specificul zonei de interes și deci în consecință înțelesul lingvistic este asociat, introducînd noțiunea de aproximație lingvistică. Pentru a ilustra această noțiune se consideră următorul exemplu:

Exemplu $\begin{cases} a \text{ este mic} \\ a \text{ și } b \text{ sînt aproximativ egale} \\ b \text{ este mai mic sau mai puțin mic} \end{cases}$

În logica fuzzy "*a este mic*" este interpretat ca asignînd un predicat fuzzy ca o valoare a variabilei ce corespunde atributului corespunzător lui *a*

$\text{înălțime}(a) = \text{mic}$

unde înălțimea este atributul implicat. În termenii unei ecuații a doua afirmație se scrie

$(\text{înălțime}(a), \text{înălțime}(b)) = \text{aproximativ egal.}$

Se va obține utilizînd regulile de mai sus

$b = LA [\text{mic} @ \text{aproximativ egal}]$

unde @ este combinarea relațiilor fuzzy dată prin

$U_{\text{mic}} @ \text{aproximativ egal}(b) = \text{Max}[U_{\text{mic}}(x) \wedge U_{\text{aproximativ egal}}(x, b)]$
cu *x* toate obiectele domeniului.

În concluzie, după Zadeh și Bellman (1976) consecința este dată de setul premiselor și depinde în mod esențial de sensul atașat seturilor fuzzy care apar în premise.

Se pot enumera principalele facilități ale logicii fuzzy Haack:

- (i) valorile de adevăr sînt fuzzy;
- (ii) setul valorilor de adevăr lingvistice nu este închis;
- (iii) valorile de adevăr sînt subiective și locale;
- (iv) validitatea poate fi caracterizată semantic;
- (v) adecvanța pentru o mare serie de aplicații.

2.5.3. Sisteme expert ce utilizează logica fuzzy

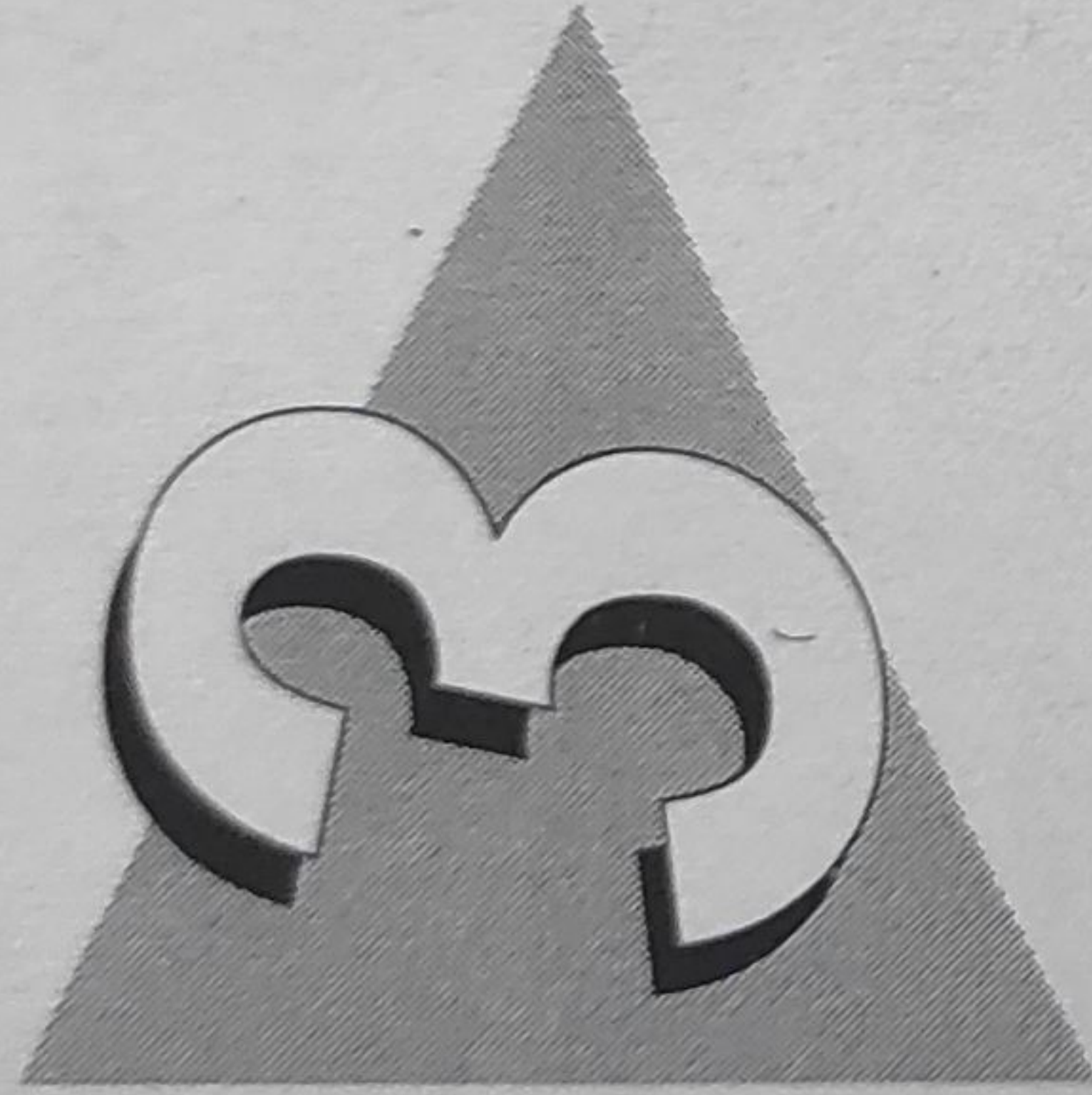
În problemele de raționare pornind de la date incerte sau incomplete sînt utilizate tehnici similare cu cele fuzzy. Astfel în MYCIN se introduce noțiunea de implicație aproximativă utilizînd numere numite factori de certitudine păstrate în baza de cunoștințe. De exemplu în regula

If the infection is primary-bacterial (and)
the site of the culture is one the sterile sites (and)
the suspected portal of entry of the organism is the gastrointestinal tract

Then there is suggestive evidence (.7) that the identity of the organism is bacteroides.

Numărul .7 este factorul de certitudine. În acest mod raționarea este vagă sau inexactă și este indicată de o valoare numerică. Operatorul pentru conjuncție în MYCIN este minimizare conform fuzzy, pe cînd disjuncția este realizată prin interpretare Bayesiană. După modul de utilizare a celor două tehnici o serie de critici pot fi aduse sistemului,

PROSPECTOR utilizează trei tipuri de relații în rețeaua inferențială: relații logice, relații plauzibile și relații contextuale. Aplicații ale teoriei fuzzy sînt găsite în relațiile logice. O serie de alte aplicații în sisteme de diagnosticare cum sînt cele dezvoltate de Saitta și Torasso, Whalen și Schott, Soula și Sanchez fac apel la logica fuzzy.



ARHITECTURI DE SISTEME EXPERT

Așa cum se deduce din prezentările făcute în capitolul 1 S.E. acoperă o clasă foarte mare de programe ce au o serie de caracteristici comune dar diferă în multe privințe față de programele clasice. Spunînd că există o separație netă între cunoștințe și mecanismul de raționare, se poate afirma că nici o cunoștință specializată nu participă la fundamentarea mecanismului de inferență. De asemenea, baza de cunoștințe are o serie de caracteristici ce oferă independența cunoștințelor, cum sînt:

- ♦ granularitatea cunoștințelor (independența acestora);
- ♦ ordinea în care sînt stocate cunoștințele în baza de cunoștințe nu influențează rezultatele;
- ♦ modificarea elementelor bazei de cunoștințe nu are implicații catastrofale asupra derulării programului;
- ♦ sistemul expert poate fi caracterizat ca o nouă modalitate de programare declarativă, programul este scris ca un ansamblu de specificații independente unele de altele, numite și elemente de cunoaștere ce sînt legate între ele dinamic printr-o procedură de inferență.

O caracteristică a sistemelor expert este cea privind competența acestora de a furniza explicații asupra raționamentelor întreprinse pentru ajungerea la rezultat, explicații ce trebuiesc să fie exprimate într-un limbaj cît mai apropiat de limbajul natural. Întrucît problemele a căror rezolvare este cerută sînt similare cu problemele pe care le rezolvă expertul, și raționamentele făcute acestora trebuiesc să corespundă. Multitudinea problemelor determină și volumul deosebit de mare al bazei de cunoștințe însă, un S.E. trebuie să fie capabil de a rezolva în aceeași măsură și probleme ce sînt afectate de cunoaștere incertă și incompletă. În astfel de situații se pot utiliza cunoștințe euristice ce permit găsirea soluției potrivite fără ca aceasta să fie nepărat soluția optimă.

Așa cum se deduce din prezentările făcute în capitolul 1 S.E. acoperă o clasă foarte mare de programe ce au o serie de caracteristici comune dar diferă în multe privințe față de programele clasice. Spunînd că există o separație netă între cunoștințe și mecanismul de raționare, se poate afirma că nici o cunoștință specializată nu participă la fundamentarea mecanismului de inferență. De asemenea, baza de cunoștințe are o serie de caracteristici ce oferă independența cunoștințelor, cum sînt:

- ◆ granularitatea cunoștințelor (independența acestora);
- ◆ ordinea în care sînt stocate cunoștințele în baza de cunoștințe nu influențează rezultatele;
- ◆ modificarea elementelor bazei de cunoștințe nu are implicații catastrofale asupra derulării programului;
- ◆ sistemul expert poate fi caracterizat ca o nouă modalitate de programare declarativă, programul este scris ca un ansamblu de specificații independente unele de altele, numite și elemente de cunoaștere ce sînt legate între ele dinamic printr-o procedură de inferență.

O caracteristică a sistemelor expert este cea privind competența acestora de a furniza explicații asupra raționamentelor întreprinse pentru ajungerea la rezultat, explicații ce trebuiesc să fie exprimate într-un limbaj cît mai apropiat de limbajul natural. Întrucît problemele a căror rezolvare este cerută sînt similare cu problemele pe care le rezolvă expertul, și raționamentele făcute acestora trebuiesc să corespundă. Multitudinea problemelor determină și volumul deosebit de mare al bazei de cunoștințe însă, un S.E. trebuie să fie capabil de a rezolva în aceeași măsură și probleme ce sînt afectate de cunoaștere incertă și incompletă. În astfel de situații se pot utiliza cunoștințe euristice ce permit găsirea soluției potrivite fără ca aceasta să fie nepărat soluția optimă.

3.1. Componentele sistemelor expert

Toate aceste caracteristici determină o structură specifică pentru sistemele expert, structură ce poate fi grupată în jurul a trei module principale, module ce determină și ceea ce se numește *sistem esențial*:

a) *Baza de cunoștințe* este reprezentată ca o structură de date ce conține ansamblul cunoștințelor specializate introduse de către expertul uman. Cunoștințele stocate în baza de cunoștințe sînt în principal descripții de obiecte în conjuncție cu relațiile dintre acestea. Baza de cunoștințe face parte din sistemul cognitiv, cunoașterea fiind memorată într-un spațiu special organizat. Forma de stocare a informației experte trebuie să asigure căutarea pieselor de cunoaștere specificate direct prin simboluri identificatoare, căutarea pieselor de cunoaștere referite indirect prin proprietăți asociate sau valori atribuite acestora, căutarea pieselor de cunoaștere prin inferențe sau lanțuri inferențiale ce pornesc de la alte piese de cunoaștere sau de la proprietăți asociate acestora, menținerea bazei de cunoștințe în concordanță cu evoluția domeniului de expertiză. În capitolul 4 se vor trata principalele modalități de reprezentare a cunoștințelor în baze de cunoștințe.

b) *Mecanismul de inferență* reprezintă noutatea sistemelor expert. El preia cunoștințele din baza de cunoștințe ce sînt utilizate pentru construirea raționamentului. Mecanismul de inferență urmărește o serie de obiective majore cum sînt: alege strategia de control funcție de problema curentă ce o are de rezolvat, elaborează planul de rezolvare a problemei după necesitate, execută comutarea de la o strategie de control la alta, execută acțiunile prevăzute în planul de rezolvare, constituie informațiile de control pentru mecanismele fundamentale ale mecanismului de inferență. Cu toate că în esență mecanismul de inferență este constituit dintr-un ansamblu de proceduri în sensul obișnuit al termenului, modul în care utilizează cunoștințele nu este prevăzut prin nici un program, acesta fiind condiționat de datele și cunoștințele pe care le posedă.

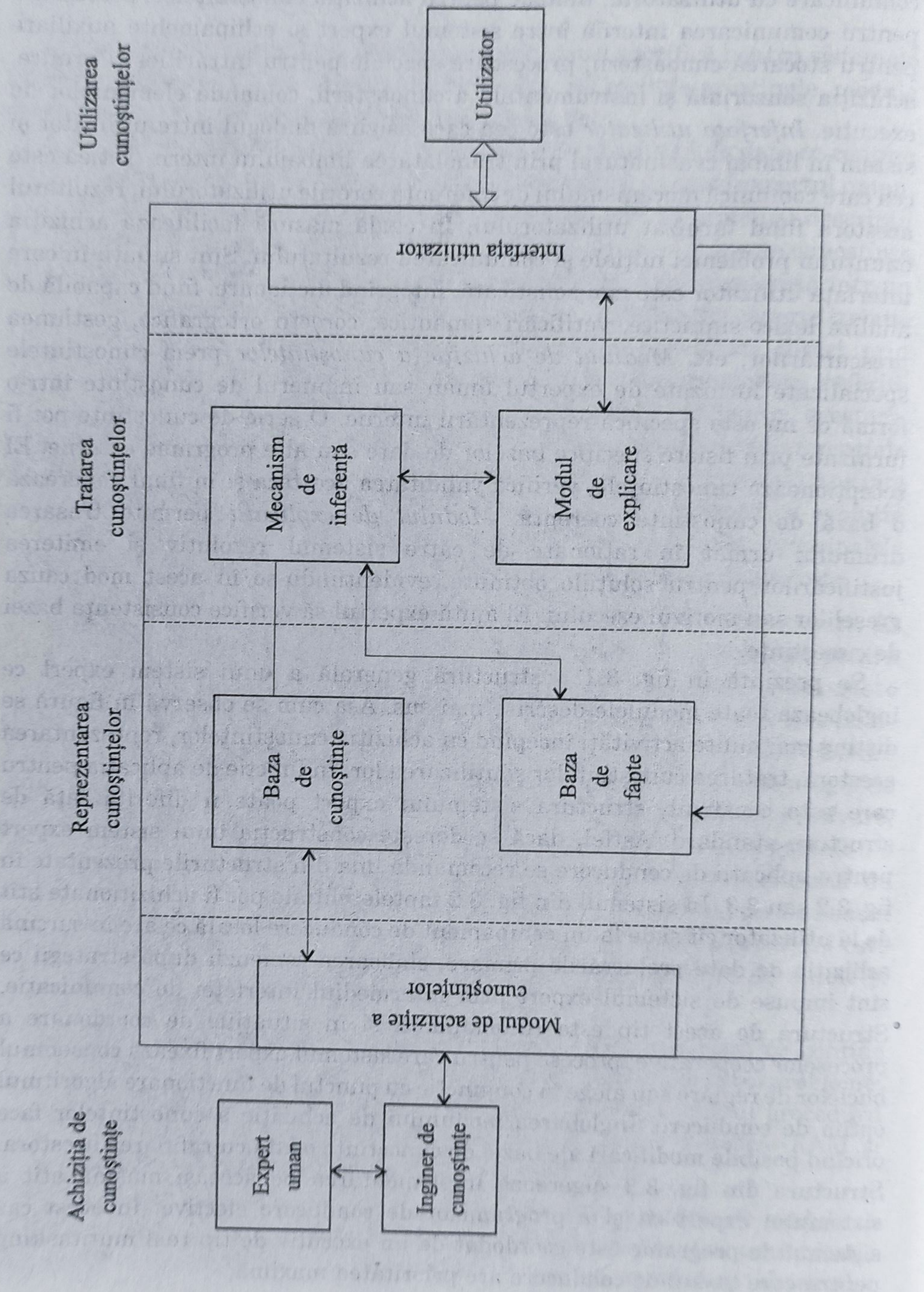
c) *Baza de fapte* este reprezentată de o memorie auxiliară ce conține toate datele utilizatorului (faptele inițiale ce descriu enunțul problemei de rezolvat), și rezultatele intermediare produse în cursul procedurii de deducție. Conținutul bazei este păstrat într-o memorie RAM și poate fi conservată doar la cererea utilizatorului.

Pe lângă aceste module un sistem expert mai conține o serie de module ce asigură comunicarea cu operatorul și expertul uman. Modulul de comunicație este destinat furnizării interfețelor specifice pentru utilizatorii sistemului

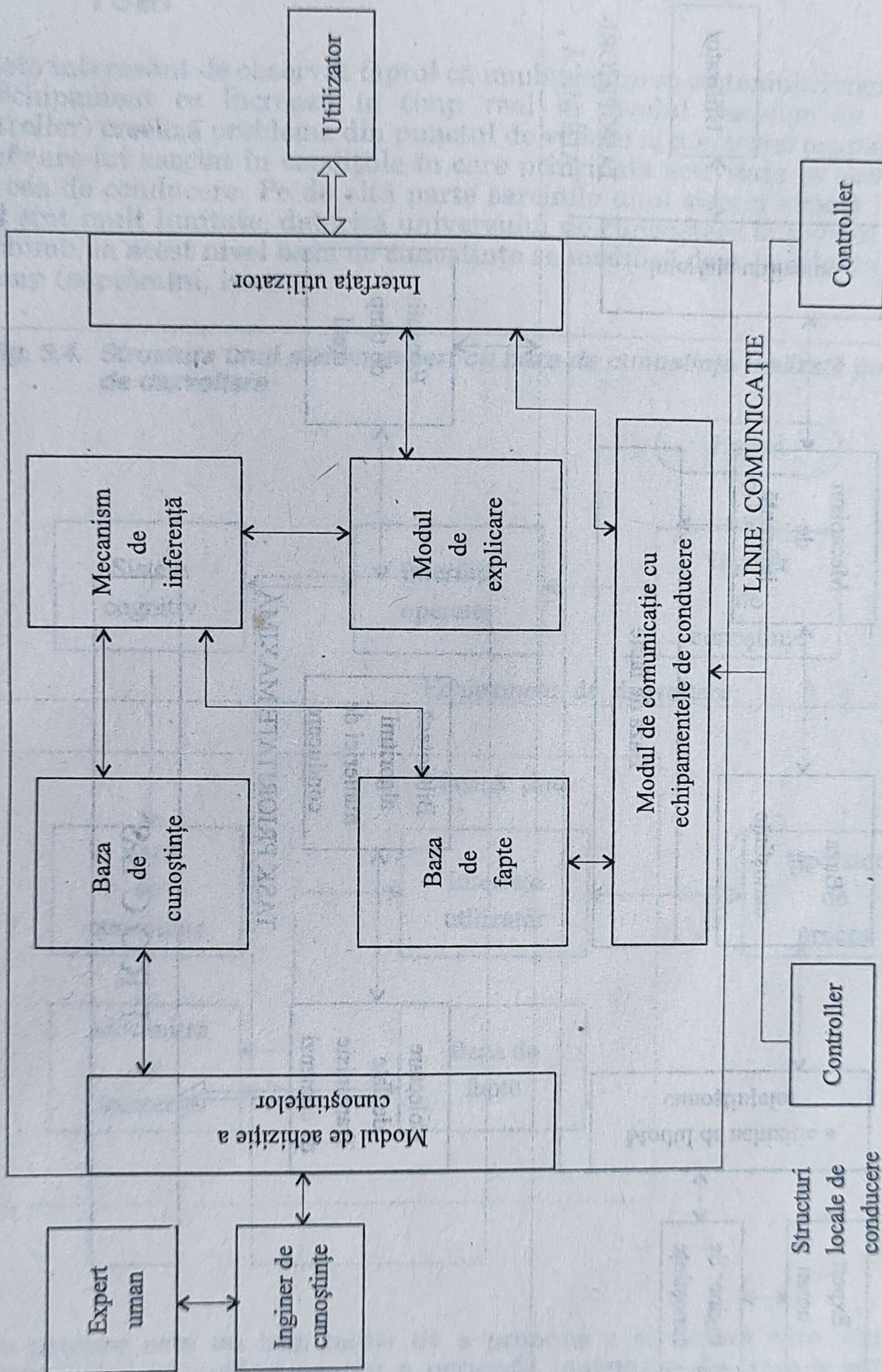
expert cât și pentru achiziția de cunoștințe. În compunerea acestui modul intră procesoare specifice pentru limbajele de comunicare înglobând limbaje de comunicare cu utilizatorul, limbaje pentru achiziția cunoștințelor, procesoare pentru comunicarea internă între sistemul expert și echipamente auxiliare pentru stocarea cunoașterii, procesoare speciale pentru intrări ieșiri grafice, achiziția senzorială și instrumentală a cunoașterii, comanda elementelor de execuție. *Inferfața utilizator* este cea care asigură dialogul între utilizator și sistem în limbaj cvasinatural prin translatarea limbajului intern. Tot ea este cea care comunică mecanismului de inferență cererile utilizatorului, rezultatul acestora fiind furnizat utilizatorului. În egală măsură facilitează achiziția enunțului problemei inițiale și comunicarea rezultatului. Sînt situații în care interfața utilizator este mai sofisticată, integrînd dicționare, fiind capabilă de analize lexico-sintactice, verificări semantice, corecții ortografice, gestiunea prescurtărilor, etc. *Modulul de achiziție a cunoștințelor* preia cunoștințele specializate furnizate de expertul uman sau inginerul de cunoștințe într-o formă ce nu este specifică reprezentării interne. O serie de cunoștințe pot fi furnizate prin fișiere specifice bazelor de date sau alte programe externe. El recepționează cunoștințele, verifică validitatea acestora și în final generează o bază de cunoștințe coerentă. *Modulul de explicații* permite trasarea drumului urmat în raționare de către sistemul rezolutiv și emiterea justificărilor pentru soluțiile obținute, evidențiindu-se în acest mod cauza greșelilor sau motivul eșecului. El ajută expertul să verifice consistența bazei de cunoștințe.

Se prezintă în fig. 3.1 o structură generală a unui sistem expert ce înglobează toate modulele descrise mai sus. Așa cum se observă în figură se disting mai multe activități începînd cu achiziția cunoștințelor, reprezentarea acestora, tratarea cunoștințelor și utilizarea lor. În funcție de aplicația pentru care este construit, structura sistemului expert poate fi diferită față de structura standard. Astfel, dacă se dorește construcția unui sistem expert pentru aplicații de conducere se recomandă una din structurile prezentate în fig. 3.2 sau 3.3. În sistemul din fig. 3.2 faptele inițiale pot fi achiziționate atît de la utilizator cât și de la un echipament de conducere locală ce are în sarcină achiziția de date prelucrările primare, elaboarea comenzii după strategii ce sînt impuse de sistemul expert prin intermediul interfeței de comunicație. Structura de acest tip este recomandată și în situațiile de coordonare a proceselor cooperative, procese pentru care sistemul expert fixează consemnul buclelor de reglare sau alege în conjuncție cu punctul de funcționare algoritmul optim de conducere. Înglobarea modulului de achiziție a cunoștințelor face oricînd posibile modificări ale bazei de cunoștințe odată cu rafinarea acestora. Structura din fig. 3.3 sugerează implementarea pe aceeași mașină atît a sistemului expert cât și a programelor de conducere efective. În acest caz sistemul de programe este coordonat de un executiv de tip real multitasking pentru care taskul de conducere are prioritatea maximă.

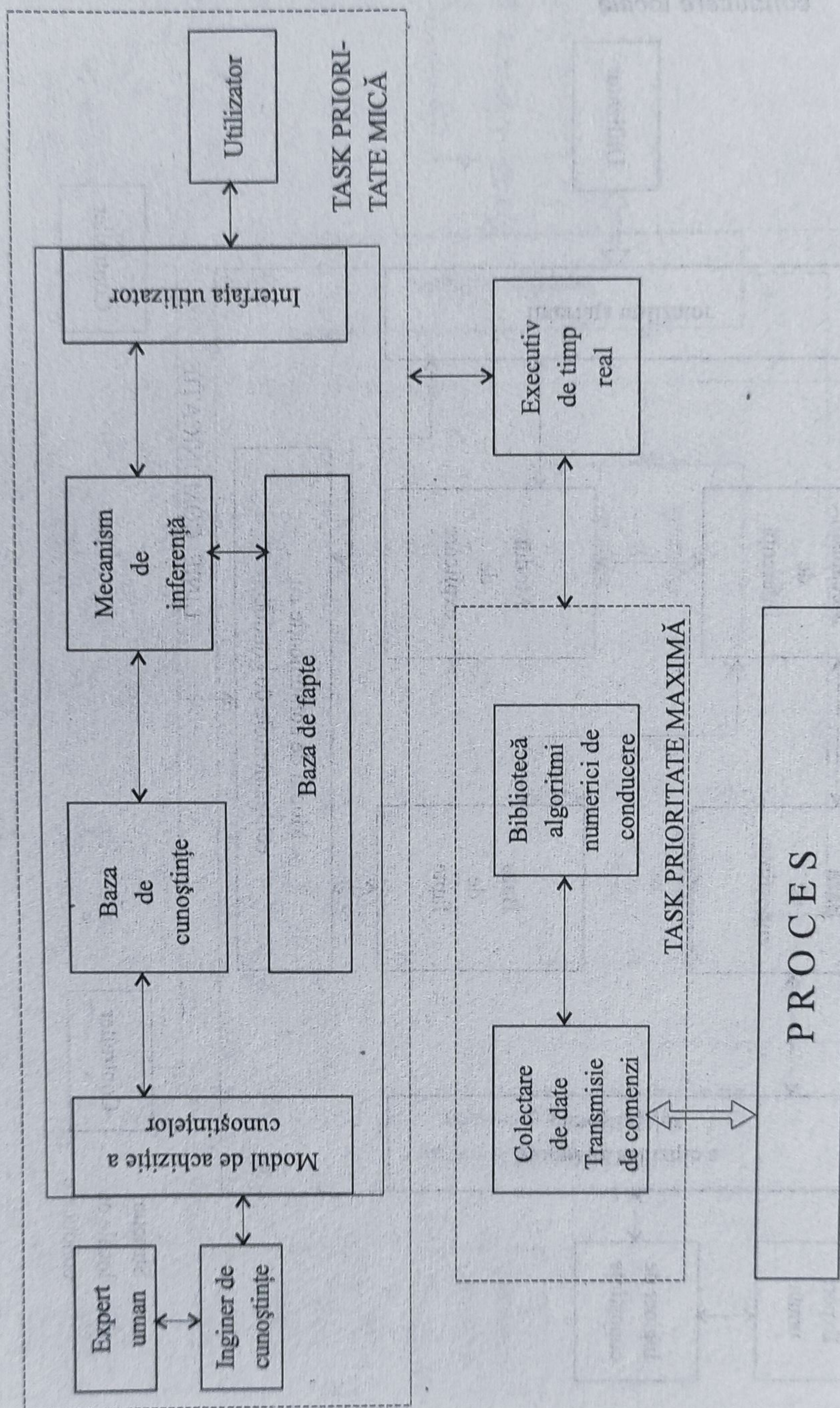
▼ Fig. 3.1. Structura generală a unui sistem expert



▼ Fig. 3.2. Structura de sistem expert în comunicație cu echipament de conducere locală



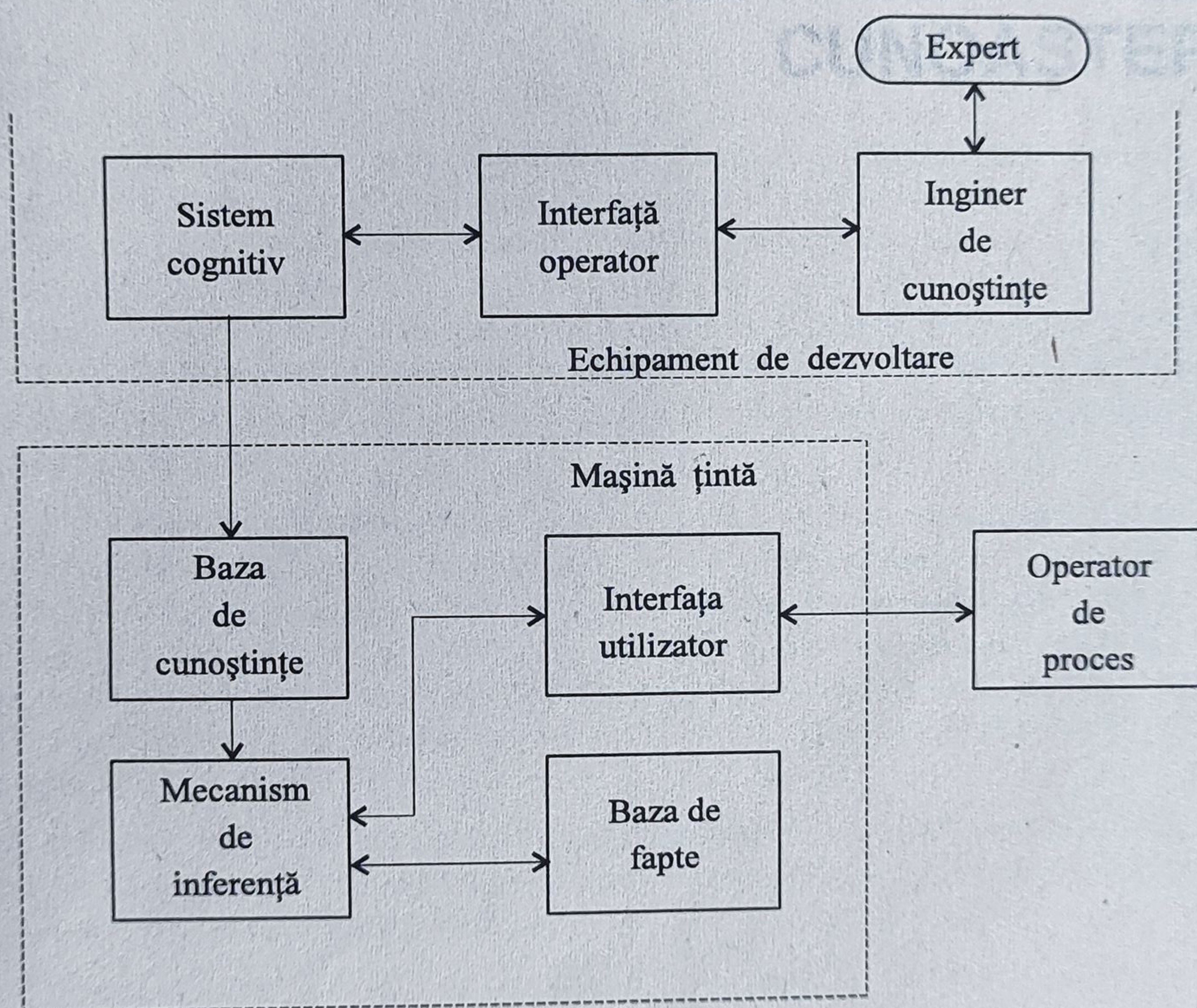
▼ Fig. 3.3. Structura de sistem expert cuplat cu algoritmi de conducere



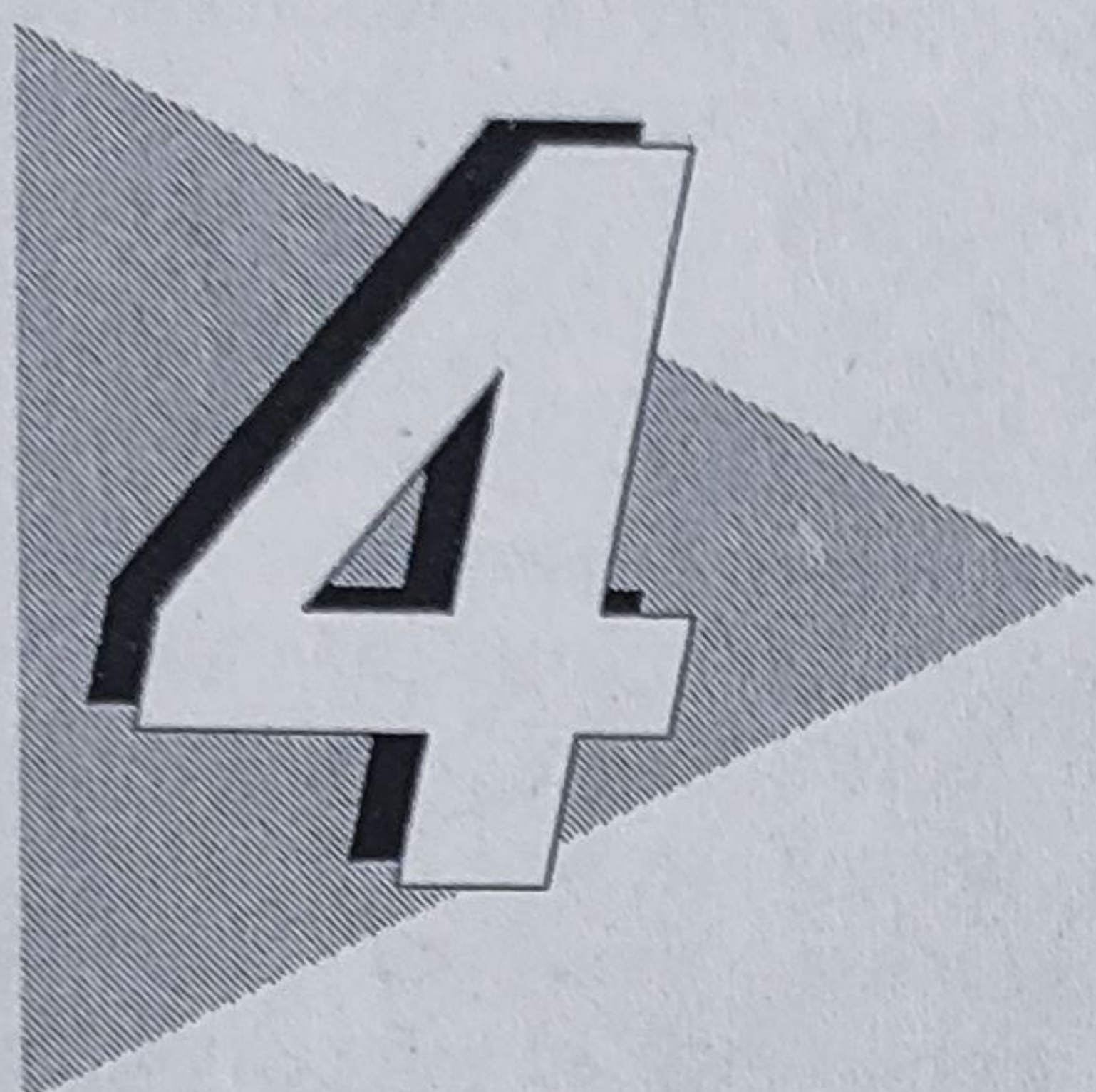
3.2. Sisteme Expert pentru aplicații în timp real

Este interesant de observat faptul că implementarea sistemului cognitiv pe un echipament ce lucrează în timp real la nivelul buclelor de reglare (controller) creează probleme din punctul de vedere al memoriei ocupate cât și al software-ului asociat în condițiile în care principala activitate la acest nivel este cea de conducere. Pe de altă parte sarcinile unui sistem expert la acest nivel sînt mult limitate, datorită universului de cunoaștere la nivelul buclei. În schimb, la acest nivel baza de cunoștințe se modifică doar la intervale mari de timp (săptămîni, luni).

▼ Fig. 3.4. Structura unui sistem expert cu baza de cunoștințe realizată pe sistem de dezvoltare



Ca urmare este un bun motiv de a propune o structură care, chiar cu inconvenientul înghețării pentru o perioadă mai mare de timp a bazei de cunoștințe oferă o mai mare suplețe sistemului (fig. 3.4). În această situație activitatea de achiziție a cunoștințelor se execută pe echipamentul de dezvoltare, baza de cunoștințe rezultată fiind transferată pe mașina țintă.



REPREZENTAREA CUNOAȘTERII

Reprezentarea cunoașterii este un calculat, construit în funcție de
conținutul informațional și de scopul urmărit. Este o reprezentare a
realității, în care se reflectă relațiile dintre obiectele și fenomenele
studiate. În acest scop, se utilizează simboluri, semne, linii, etc.
Descrierea este un proces de prelucrare a informației, care are ca scop
obținerea unei reprezentări a realității, care să fie ușor de înțeles și
de folosit. Aceasta este o activitate intelectuală, care implică gândirea
și creativitatea. Reprezentarea cunoașterii este un proces complex, care
implică multiple etape și metode.

Procesul de reprezentare a cunoașterii este un proces complex, care
implică multiple etape și metode. Este un proces de prelucrare a
informației, care are ca scop obținerea unei reprezentări a realității,
care să fie ușor de înțeles și de folosit. Aceasta este o activitate
intelectuală, care implică gândirea și creativitatea. Reprezentarea
cunoașterii este un proces complex, care implică multiple etape și
metode.

Procesul de reprezentare a cunoașterii este un proces complex, care
implică multiple etape și metode. Este un proces de prelucrare a
informației, care are ca scop obținerea unei reprezentări a realității,
care să fie ușor de înțeles și de folosit. Aceasta este o activitate
intelectuală, care implică gândirea și creativitatea. Reprezentarea
cunoașterii este un proces complex, care implică multiple etape și
metode.

Procesul de reprezentare a cunoașterii este un proces complex, care
implică multiple etape și metode. Este un proces de prelucrare a
informației, care are ca scop obținerea unei reprezentări a realității,
care să fie ușor de înțeles și de folosit. Aceasta este o activitate
intelectuală, care implică gândirea și creativitatea. Reprezentarea
cunoașterii este un proces complex, care implică multiple etape și
metode.

Procesul de reprezentare a cunoașterii este un proces complex, care
implică multiple etape și metode. Este un proces de prelucrare a
informației, care are ca scop obținerea unei reprezentări a realității,
care să fie ușor de înțeles și de folosit. Aceasta este o activitate
intelectuală, care implică gândirea și creativitatea. Reprezentarea
cunoașterii este un proces complex, care implică multiple etape și
metode.

Reprezentarea cunoștințelor într-un calculator constă în găsirea unei corespondențe între lumea exterioară și sistemul simbolic ce permite execuția raționamentelor. În acest scop se extrag din observațiile făcute asupra obiectelor, a faptelor și fenomenelor, acele caracteristici cărora li se pot asocia semnificații determinate în imaginea mentală formată despre lume. Descrierea are ca scop deosebirea imaginii obiectului dat de imaginile celorlalte obiecte ale lumii înconjurătoare la fel cum obiectul fizic se deosebește de restul obiectelor lumii reale. Aceasta arată că descrierea reflectă însăși cunoașterea despre acel obiect.

Filozofic cunoașterea poate fi definită ca o reflectare activă în conștiință a lumii reale, a esențialului și generalului din fenomene. Pot fi reliefate două componente ale cunoașterii științifice și anume: *cunoașterea reflexivă* bazată pe reflectarea exterioară a faptului științific și *cunoașterea generativă* bazată pe crearea de noi obiecte abstracte, prin constituirea de momente inițiale pentru procesele de construire a unor noi fapte științifice și programe de acțiune.

Cunoașterea este raportată la un "subiect cunoscător", adică un agent ce are capacitatea de interpretare a unei informații pe care o are la dispoziție prin extragere din memoria proprie.

Cunoașterea este *empirică* dacă informațiile despre obiecte, fapte, fenomene, procese, necunoscute este sesizată de subiectul cunoscător prin organele sale senzoriale sau prin intermediul unor aparate sau instrumente.

Cunoașterea este *teoretică* dacă se desfășoară după raționamente și judecăți reliefând legăturile interne, cauzalitatea, legitățile după care se dezvoltă structurile și se desfășoară procesele. Ea se dezvoltă din cunoașterea empirică, prin analiză, sinteză, deducție, inducție, generalizare, particularizare.

Datorită caracterului de reflectare în conștiință a realității, cunoașterea are un caracter individual, subiecții cunoscători pot avea nivele de cunoaștere diferită funcție de lucrurile reprezentate. Termenul de cunoaștere arată că de fapt reprezintă o cunoaștere *parțială* dacă permite deosebirea părții reprezentate de restul lumii, respectiv cunoaștere *incompletă* dacă apar situații în care părți distincte ale lumii pot avea aceeași reprezentare.

În transpunerea de la subiecții umani la subiecții cunoscători de tip "program pentru calculator" deci de la psihologia gândirii la inteligența artificială conceptul de cunoaștere își conservă calitățile de mai sus. În vederea utilizării unui program cunoașterea este memorată sub forma unor piese de cunoaștere ce descriu fapte, fenomene, procese, evenimente dintr-o parte a lumii reale ce constituie domeniul de competență al programului inteligent. Piese de cunoaștere alcătuiesc un model al lumii la care programul are acces prin intermediul procedurilor de organizare, clasificare, căutare și recunoaștere.

Se poate, în final, defini un sistem cognitiv ca totalitatea pieselor de cunoaștere, a modului de stocare și a procedurilor de acces la acestea.

Problema fundamentală a inteligenței artificiale este cea de definire a unor metode pentru reprezentarea unei mari cantități de cunoștințe într-o formă ce permite stocarea și utilizarea eficientă a acestora.

4.1. Noțiuni introductive despre reprezentarea cunoașterii

Reprezentarea cunoașterii poate fi conform celor de mai sus considerată ca o relație de definiție $A \Leftrightarrow B$ ce stabilește o legătură între un simbol identificator A (numele entității reprezentate) și o expresie B formulată în limbajul de reprezentare al sistemului, ce descrie caracteristicile entității și structura sa cu ajutorul unor termeni primari, termeni purtători de semnificație ai limbajului. Expresia B este o reprezentare a entității A sau a piesei de cunoaștere A în sistemul cognitiv considerat ca gazdă a reprezentării cunoașterii.

Orice activitate de reprezentare a cunoașterii are loc după stabilirea ordinii conceptuale pe baza căreia se construiește metoda efectivă de reprezentare, ce are în vedere stabilirea problemelor relative la piesele de cunoaștere ca obiecte primitive ale sistemului formal, probleme ce pot fi sintetizate ca mai jos:

- ♦ *stabilirea sistemului de meta-reprezentare* ce are ca obiectiv asigurarea unor modalități adecvate de reprezentare a cunoașterii despre modelul lumii reale cât și pentru componentele proprii ale sistemului inteligent;
- ♦ *stabilirea sistemului de clasificare* ce urmărește gruparea în clase, precum și ordonarea acestora după criterii ce rezultă din analiza

relevanței proprietăților ce sînt folosite pentru caracterizarea obiectelor ce formează piesele de cunoaștere;

- ♦ stabilirea *sistemului de organizare* privitor la gruparea și ordonarea elementelor și claselor, avînd în vedere asigurarea realizării unor procese fundamentale cum sînt: acces, căutare, corespondență, substituie, inferență deductivă, inferență inductivă, grupare și regrupare, organizare și reorganizare, protecție.

Se poate apoi trece la enunțuri văzute ca obiecte ale limbajului formal. Sînt necesare reguli privind formarea enunțurilor din piese de cunoaștere, stabilirea sistemului teoretic prin definirea unui set de enunțuri-axiomă și a regulilor de inferență. Se prezintă mai jos principalele probleme relative la piesele de cunoaștere.

4.1.1. Sistemul de meta-reprezentare

Prin meta-reprezentare se înțelege acea reprezentare a cunoașterii despre cunoașterea însăși și despre reprezentările ei. Ca urmare domeniul de competență al meta-reprezentării este chiar *cunoașterea abstractă*. Principalele piese de meta-cunoaștere sînt:

- ♦ *meta-cunoașterea reprezentării obiectelor* ce cuprinde descrierea schemelor de reprezentare utilizate pentru piesele de cunoaștere de tip obiectual;
- ♦ *meta-cunoașterea reprezentării funcțiilor* ce cuprinde descrierea șabloanelor funcționale destinate pieselor de cunoaștere de tip procedural;
- ♦ *meta-cunoașterea reprezentării relațiilor* ce cuprinde descrierea schemelor de reprezentare a pieselor de cunoaștere, reprezentînd relații între piese de cunoaștere de tip obiectual;
- ♦ *meta-cunoașterea reprezentării regulilor de inferență* ce cuprinde descrierea schemelor de reprezentare a regulilor;
- ♦ *meta-cunoașterea reprezentării strategiilor de raționamente* ce cuprinde descrierea meta-regulilor și a formalismelor de reprezentare a planurilor;
- ♦ *meta-cunoașterea auto-reprezentării* ce cuprinde descrierea sistemului inteligent în el însuși.

Se disting situații specifice funcție de modul în care se dispune de informația necesară pentru reprezentarea expresiei definitorii a piesei de cunoaștere, fapt ce determină meta-cunoașterea corespunzătoare:

- ♦ *reprezentarea descriptivă* realizată prin caracterizarea directă a componentelor, structurii și proprietăților piesei de cunoaștere;

- ♦ *reprezentarea comparativă* în care pentru a se reprezenta piesa de cunoaștere *A* expresia *B* conține o referință la o piesă auxiliară numită și prototip și o informație ce marchează diferențele între aceasta și piesa prototip. Astfel pentru reprezentarea piesei de cunoaștere *A* se reprezintă o referință *B* la o unitate auxiliară *C* și o informație suplimentară *D*, care punctează diferențele între *A* și *C*;
- ♦ *reprezentarea constructivă* (generativă) în care expresia definitorie cuprinde o relație generatoare a elementelor componente, structurii și proprietății piesei de cunoaștere;
- ♦ *reprezentarea enumerativă* în care expresia definitorie cuprinde referiri la elementele unei mulțimi care este definită ca piesă de cunoaștere.

Ca strategie generală se poate afirma că în general se optează pentru una din reprezentări, dar o reprezentare completă este oferită doar printr-o combinație finită a metodelor de reprezentare, deoarece cunoașterea despre un obiect este dată parțial în termeni descriptivi, parțial în termeni comparativi față de un obiect cunoscut, parțial în termeni generativi și enumerativi. Este de dorit ca sistemul să accepte completarea reprezentării folosind mecanismele inferențiale asociate sistemului cognitiv.

4.1.2. Sistemul de clasificare

Ordinea conceptuală se realizează cu ajutorul unui sistem de clasificare ce urmărește stabilirea unui număr finit de clasificări a pieselor de cunoaștere care să constituie ipostaze diferite ale sistemului de interpretare. Clasificarea, presupune stabilirea unor grupări numite clase, pe baza unei similarități denumită și criteriu de clasificare, proces ce se efectuează în două etape:

- ♦ *clasificare conceptuală* prin care proprietățile obiectului sînt analizate din punct de vedere al relevanței în raport cu obiectele utilitare ale sistemului, pentru a fi grupate apoi în criterii;
- ♦ *clasificare instanțială* prin care piesele de cunoaștere sînt plasate în clase potrivit criteriului de clasificare stabilit pentru fiecare piesă de cunoaștere.

În general, prin criteriu de clasificare se înțelege stabilirea categoriilor logice, a proprietăților ce determină pe mulțimea pieselor de cunoaștere o partajare în clase. Stabilirea criteriilor de clasificare reprezintă o fază conceptuală de mare importanță în reprezentarea cunoașterii. Clasificarea este orientată către obiectivul utilitar al aplicației. Astfel pentru un sistem expert independent de domeniul de aplicație limbajul de reprezentare a cunoașterii trebuie să permită comunicarea criteriilor de clasificare ca piese de cunoaștere privind sistematizarea bazei de cunoștințe în funcție de obiectivele generale ale sistemului. O clasă este constituită prin predicție,

elementele sale determină că predicatul ce semnifică proprietățile din criteriul de clasificare să ia valoarea adevărat.

Se spune despre o clasă că este o *extensiune* a unui predicat a cărui instanțiere este proprietatea luată drept criteriu de clasificare. Un criteriu de clasificare care nu poate decide satisfacerea condițiilor de unicitate a clasificării este un criteriu de grupare sau *clustering*. Ca urmare un element poate să aparțină mai multor clase. Ca exemplu, piesa de cunoaștere reprezentând automobilul DACIA aparține unei singure clase și anume clasa autovehiculelor. În același timp aparține mai multor grupări cum sînt: grupul autoturismelor, grupul autovehiculelor cu motor Otto, grupul autovehiculelor de fabricație românească, etc. Prin apartenența sa la clasa autovehiculelor DACIA moștenește acele proprietăți ce au servit drept criteriu al clasificării sale ca automobil: destinație - transport, consumă - combustibil, are sistem de rulare, etc. Atașarea la grupări nu modifică proprietățile ce îl definesc ca membru al clasei ci asigură o serie de noi trăsături de importanță secundară a căror influență asupra procesului de interpretare sînt specificate prin procedeele de atașare procedurală.

4.1.3. Sistemul de organizare

Prin acest sistem se înțelege activitatea de ordonare și grupare a pieselor de cunoaștere după criterii funcționale ce rezultă din analiza modalităților de preluare din baza de cunoștințe a unor piese de cunoaștere necesare unor procese în sistemul rezolutiv. În general această activitate nu dispune de abordări teoretice care să permită fundamentarea de soluții ce se vor utiliza în aplicații practice. În diferite programe aplicative s-au utilizat soluții proprii dependente de domeniul de competență al aplicației. Se pot enumera cîteva dintre principalele criterii conceptuale de organizare:

- ♦ accesul la piesele de cunoaștere și la componentele lor;
- ♦ modul de execuție a proceselor inferențiale;
- ♦ existența relațiilor de ordine;
- ♦ protecția împotriva manevrelor neautorizate;
- ♦ asigurarea consistenței bazei de cunoștințe.

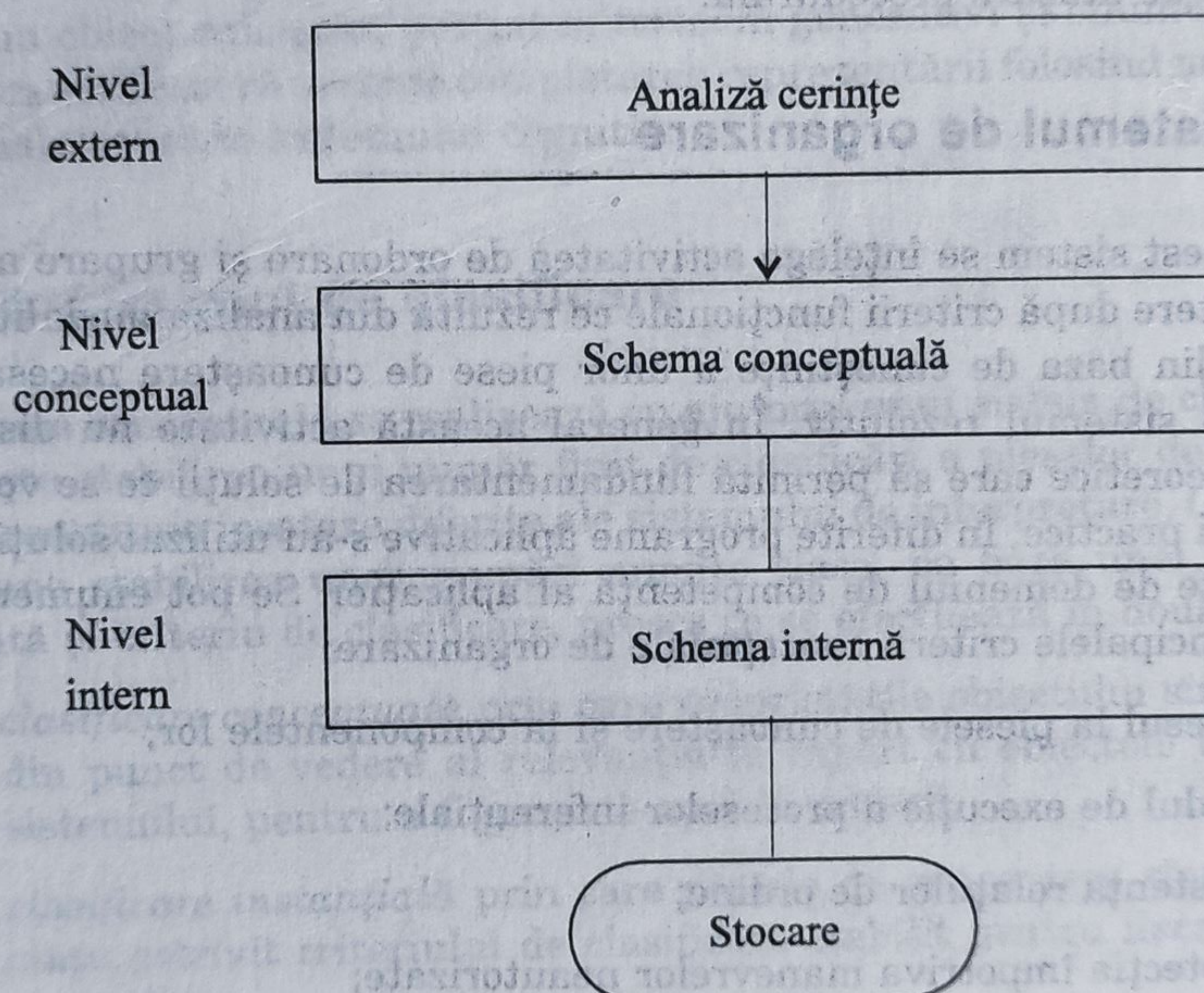
Față de programele scrise în limbajele de programare ce utilizează metode de organizare și acces standard, limbajale destinate inteligenței artificiale oferă extrem de puține facilități de organizare a bazelor de cunoștințe, lăsînd în general această sarcină în seama realizatorilor de aplicații. Este poate unul din motivele pentru care aplicațiile de inteligență artificială nu au cunoscut pînă în prezent o dezvoltare asemănătoare aplicațiilor în tehnicile tradiționale. Se poate remarca faptul că procedeele specifice structurilor de date în programarea clasică nu sînt disponibile aici.

4.2. Metode de reprezentarea cunoașterii

Reprezentarea cunoașterii urmărește descrierea universului în care sistemul efectuează raționamente sub formă de entități corespunzătoare indivizilor și sub formă de simboluri pentru relațiile între aceștia. Deoarece în timp cunoașterea despre indivizi și relațiile dintre aceștia poate să se modifice nu este posibilă o reprezentare declarativă a stării inițiale și se pune problema înregistrării enumerative a succesiunilor de stare cât și a mecanismului ce asigură transformarea stării inițiale spre stările succesive.

O arhitectură de reprezentare a cunoștințelor poate fi considerată ca fiind constituită pe trei nivele. Cele trei nivele sînt stabilite de locul în care se judecă cunoștințele. O schemă de reprezentare a cunoștințelor pe trei nivele este reprezentată în fig. 4.1.

▼ Fig. 4.1. Arhitectura pe trei nivele de reprezentarea datelor



Nivelul intern este constituit din schema internă ce descrie structura de stocare fizică a cunoștințelor în baza de cunoștințe. La acest nivel se descriu detaliile complete ale stocării, precum și modul de acces la cunoștințe.

Nivelul conceptual sau schema conceptuală descrie structura întregii baze de cunoștințe pentru o comunitate de utilizatori. La nivelul conceptual se face o descriere completă a bazei de cunoștințe, ascunzînd detaliile legate de stocarea fizică, concentrîndu-se asupra descrierii entităților, tipurilor de date, relațiilor dintre ele, precum și a restricțiilor asociate. Poate fi utilizat cu bune rezultate la acest nivel un model de nivel înalt, sau un model specific de implementare.

Nivelul extern sau nivelul vizual (utilizator), include o colecție de scheme externe ce descrie baza de cunoștințe prin prisma diferiților utilizatori. Fiecare grup utilizator descrie baza de cunoștințe prin prisma propriilor interese. Există tendința la acest nivel ca grupuri de utilizatori să ascundă detalii de care nu sînt interesate. Și la acest nivel se pot folosi modele de implementare sau modele de nivel înalt.

Desigur că, în multe sisteme nu există o distincție netă între cele trei nivele. Cu toate acestea se poate remarca în majoritatea sistemelor cognitive un nivel conceptual puternic ce suplinește aparent de cele mai multe ori celelalte nivele. De asemenea, se remarcă o contopire în aplicații între nivelele conceptual și extern. În plus arhitectura pe trei nivele reprezintă numai o descriere a datelor ce vor forma baza de cunoștințe. Grupurile de utilizatori se referă numai la schema externă, deci, sistemul cognitiv este cel ce va trebui să transforme schema externă în schema conceptuală.

Unul de conceptele de bază de reprezentare a cunoștințelor este cel de entitate. O entitate este un obiect al lumii reale, cu o existență independentă. O entitate este un obiect cu existență fizică, persoană particulară, automobil, companie, activitate, curs universitar, etc. Orice entitate are o serie de proprietăți numite attribute ce particularizează entitatea respectivă. De exemplu, pentru o entitate automobil se pot enumera o serie de attribute cum sînt marca, combustibilul utilizat, capacitate cilindrică, etc. Valorile acestor attribute au ca scop identificarea entității.

Unele attribute pot fi împărțite în părți mai mici cu semnificație independentă. Un astfel de atribut este un atribut *complex*. Un exemplu este cel al atributului adresă, ce poate fi subdivizat în attributele componente Oraș, Județ, Cod poștal, Stradă. La rîndul său atributul Stradă este definit prin attributele Nume stradă, Număr, Scară, Număr apartament. Un alt exemplu este cel al atributului Nume pentru o persoană fizică ce poate fi subdivizat în attributele simple Nume, Inițială, Prenume. Attributele ce nu sînt compuse se numesc attribute *atomice*. Valoarea atributelor compuse se formează prin compunerea valorilor atributelor atomice.

Multe attribute au valoare unică pentru o entitate particulară și sînt numite attribute cu o *singură valoare*. Spre exemplu vîrsta unei persoane. Există attribute ce pot lua mai multe valori ca de exemplu gradele didactice într-o universitate, culoarea unui automobil, etc. Acestea sînt attribute cu *mai multe valori*.

Atributele derivate sînt attributele ce se pot determina din relațiile ce există între ele ca de exemplu, vîrsta unei persoane ce se poate determina din data curentă și data de naștere. În anumite situații o entitate particulară poate să nu aibă valori aplicabile pentru toate attributele, ca de exemplu în cadrul adresei numărul apartamentului cînd este vorba de o casă particulară.

Descrierea atributelor entității este numită *schema entității* și specifică o structură comună fixată a entităților de același tip. Setul instanțelor entităților individuale la un anumit moment este numită extensie a entității.

Fiecare atribut al unei entități tip are asociat un set de valori, numit și *domeniu*, ce specifică valorile posibile pe care le poate lua. De exemplu, vârsta unui angajat este cuprinsă între 16 și 70 ani. Matematic, atributul A al entității tip E poate fi definit ca o funcție de la E la setul valorilor posibile ale lui V , sau la toate subseturile lui V .

$$A: E \rightarrow P(V)$$

Valoarea atributului A pentru entitatea e , se va nota ca $A(e)$. Pentru un atribut simplu $A(e)$ este o valoare cu un singur element, un atribut null nu are valoare sau altfel spus are valoarea null. Pentru un atribut compus A setul de valori este format ca produsul cartezian din $P(V_1), P(V_2), \dots, P(V_n)$ unde V_1, V_2, \dots, V_n reprezintă setul valorilor unui simplu component al lui A . Deci

$$V = P(V_1) \times P(V_2) \times \dots \times P(V_n)$$

Între entități se pot stabili o serie de relații ce pot avea la rîndul lor atribute ce le caracterizează. Dacă valoarea unui atribut este determinată prin combinarea entităților participante într-o relație instantă, și nu prin entități singure, atunci atributul se va specifica ca un *atribut relație*.

Se pot distinge pentru aplicațiile de inteligență artificială următoarele clase de metode de reprezentare:

- ♦ *metode logice* - sînt acele metode ce privesc cunoașterea ca o serie de aserțiuni (enunțuri adevărate) privind cunoștințele și relațiile între ele. Metoda permite folosirea regulilor de inferență direct asupra pieselor de cunoaștere din baza de cunoștințe. Are însă dezavantaje prin soluțiile nesatisfăcătoare de sistematizare a bazei de cunoștințe, prin inadecvanța reprezentării cunoașterii despre acțiuni, precum și a reprezentării regulilor euristice;
- ♦ *metode relaționale* - sînt metodele prin care cunoașterea este reprezentată pornind de la relațiile între obiecte sub formă de grafuri și rețele. Metodele din această categorie permit organizarea cunoștințelor funcție de omogenitatea acestora ce conduc la clase și sorturi. Metodele relaționale reprezintă de regulă legăturile dintre obiectele universului ca pe niște elemente ale cunoașterii, spre deosebire de metodele logice în care acestea de regulă se deduc;
- ♦ *metode procedurale* - în care cunoașterea este reprezentată sub forma de proceduri ce permit obținerea stărilor la momentele specificate pornind de la stările inițiale sau intermediare.

Descrierea formală a unui concept în termenii limbajului de reprezentare, prin care se diferențiază de alte concepte sau prin care se poate aprecia echivalența cu alte concepte similare poartă numele de *definiție*. Definiția are ca obiectivitate modelarea obiectelor și a relațiilor dintre ele astfel încît să fie specificate caracteristicile esențiale ale acestora cît și posibilitatea de a facilita operațiile în care obiectele și relațiile sînt frecvent implicate. Este normal ca pentru un concept să poată fi date mai multe definiții, fiecare definiție fiind o descriere parțială dependentă de perspectiva din care este privită.

Altă informație privind un concept este dată de relațiile acestuia cu entități de tip *acțiune*. În general un obiect poate să ocupe în cadrul unei acțiuni poziția de *argument*, deci asupra căruia se acționează cât și poziția de *rezultat* al acțiunii. Se numesc *cazuri* acele participări ale obiectelor la acțiuni ce se deosebesc prin semnificație. Pentru un concept dat este relevant să se specifice o listă a acțiunilor în care conceptul reprezentat se constituie ca argument și o listă a acțiunilor în care acesta se constituie ca rezultat. Pe lângă posibilitatea efectuării de raționamente se pot genera obiecte instanță ale conceptului prin efectuarea operațiilor ce apar în lista atașată poziției de rezultat pentru acel concept.

Se va numi *generalizare* a unui concept C o definiție mai puțin restrictivă a acestuia D , în care orice instanță a conceptului C este și instanță a conceptului D , concept mai general. Generalizarea face ca din definiția conceptului C să se omită unele proprietăți mai puțin relevante. O serie de proprietăți sînt exprimate printr-un predicat $P(x, C, D)$ în care x este o instanță de concept, care dacă satisface și predicatul pentru generalizare va satisface și definiția conceptului C .

$$DEF(x, D) \wedge P(x, D, C) \rightarrow DEF(x, C)$$

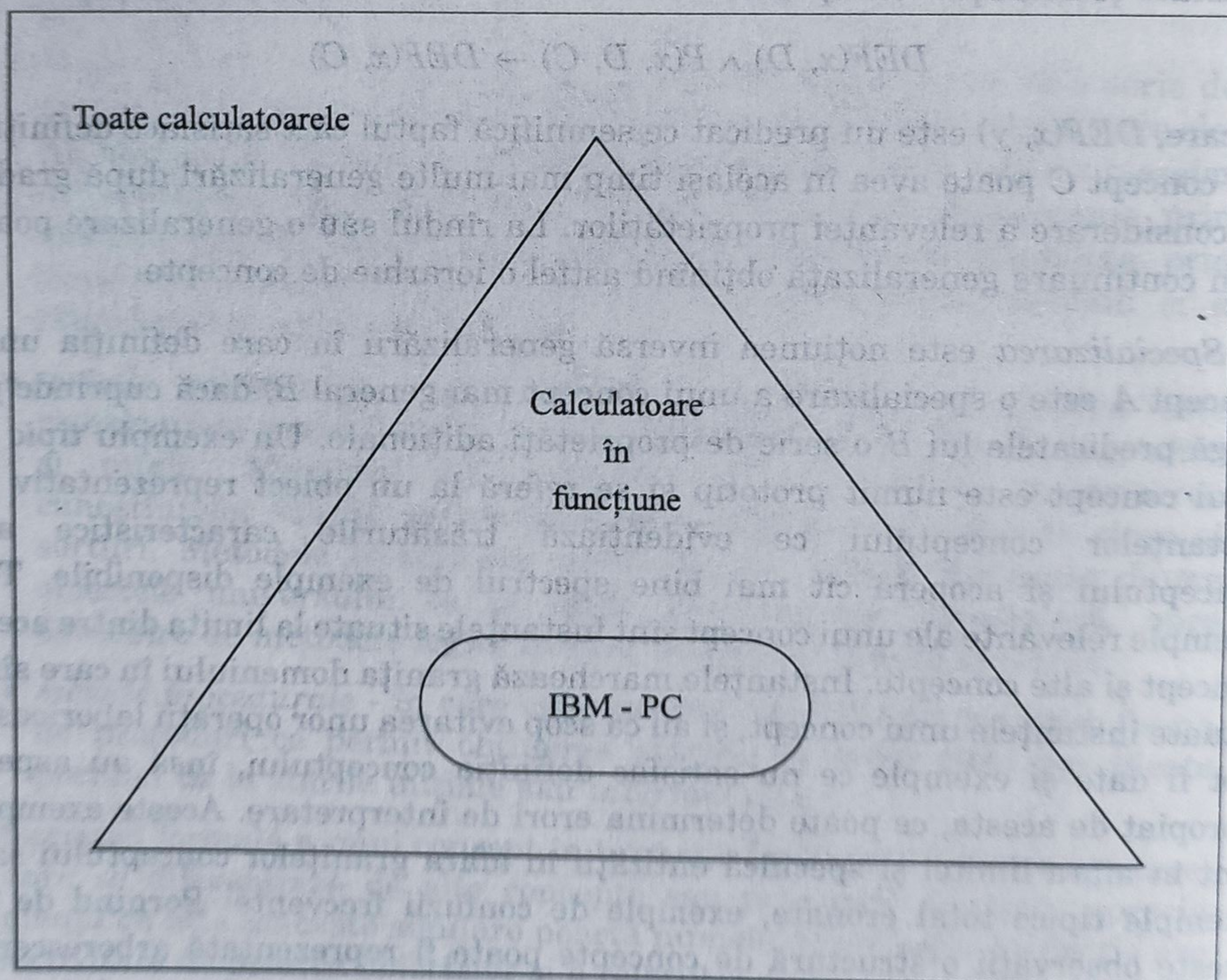
în care, $DEF(x, y)$ este un predicat ce semnifică faptul că x satisface definiția. Un concept C poate avea în același timp mai multe generalizări după gradul de considerare a relevanței proprietăților. La rîndul său o generalizare poate fi în continuare generalizată obținînd astfel o ierarhie de concepte.

Specializarea este noțiunea inversă generalizării în care definiția unui concept A este o specializare a unui concept mai general B , dacă cuprinde pe lângă predicatele lui B o serie de proprietăți adiționale. Un exemplu tipic al unui concept este numit prototip și se referă la un obiect reprezentativ al instanțelor conceptului ce evidențiază trăsăturile caracteristice ale conceptului și acoperă cât mai bine spectrul de exemple disponibile. Tot exemple relevante ale unui concept sînt instanțele situate la limita dintre acest concept și alte concepte. Instanțele marchează granița domeniului în care sînt situate instanțele unui concept, și au ca scop evitarea unor operații laborioase. Pot fi date și exemple ce nu satisfac definiția conceptului, însă au aspect apropiat de acesta, ce poate determina erori de interpretare. Aceste exemple sînt în afara limitei și specifică entității în afara granițelor conceptului sau exemple tipice total eronate, exemple de confuzii frecvente. Pornind de la aceste observații o structură de concepte poate fi reprezentată arborescent, înaintarea spre frunză reprezentînd o specializare, pe cînd apropierea de rădăcină o generalizare a conceptului.

4.2.1. Reprezentarea cunoașterii în limbajul calculului cu predicate de ordinul întâi

În acest mod de reprezentare piesele de cunoaștere sînt descrise prin expresii ale căror componente sînt formule ale limbajului. Metoda de reprezentare are avantajul introducerii pieselor de cunoaștere direct în sistemul rezolutiv dacă acesta este conceput pe baza unor mecanisme definite în calculul cu predicate de ordinul întâi. Baza reprezentării este dată de caracteristica de funcție propozițională a predicatului. O piesă de cunoaștere exprimată în limbaj natural este descompusă în propoziții elementare adecvate numite "*asertiuni*" ce specifică fapte (proprietăți, relații legate de piesa de cunoaștere). O propoziție elementară este generată de un predicat cu un număr finit de locuri în care se specifică variabile formale sau obiecte în mulțimea suport.

▼ Fig. 4.2. Plasarea obiectelor în univers



Se poate spune că acest mod de reprezentare este constituit din două etape și anume:

- ♦ *reprezentarea propozițională* - piesele de cunoaștere sînt descompuse în aserțiuni legate prin conectivele logice specifice calculului propozițional;
- ♦ *reprezentarea predicativă* - în care fiecare aserțiune este descompusă în componentele sale, predicate sau obiecte.

Calculul cu predicate reprezintă un model al raționării cu două valori, model ce implică analiza conexiunilor între clauze. Modul de raționare este asemănător raționării umane. Fie următoarea inferență simplă:

Toate calculatoarele sînt în funcțiune; (i)

IMB PC este un calculator; (ii)

Deci, IMB PC este în funcțiune (iii)

Inferențele în calculul propozițional rezultă din structura formulării. O propoziție atomică este cea mai scurtă propoziție cu înțeles în calculul propozițional. Oricum, inferența din formularea (i) necesită analiza propozițiilor primare. În special relația subiect-predicat trebuie luată în considerație. Calculul cu predicate este o extensie a calculului propozițional și constă în faptul că acesta extinde forma propozițională.

Pentru descrierea calculului cu predicate sînt necesare noi definiții și notații. Un termen este o variabilă sau o constantă, dată prin nume. Variabila x , unde x semnifică un termen este un termen. În (ii) constanta IMB PC reprezintă o instanță a lui x , adică un tip particular de calculator. Ori de cîte ori o variabilă este egală cu o constantă se spune că variabila este instanțiată. Notăția $CALCULATOR(y)$ indică faptul că y este un calculator, iar notația $FUNCTIONEAZĂ(x)$ indică proprietatea lui x de a fi în funcțiune. Ambele $CALCULATOR(y)$ și $FUNCTIONEAZĂ(x)$ sînt numite predicate și au aritatea 1, întrucît fiecare indică un singur argument. În general predicatele sînt relații cu aritate n , indicînd prin aceasta faptul că ele conțin n argumente. Atunci cînd toate variabilele sau argumentele predicatului sînt instanțiate, expresia rezultantă este o propoziție a calculului propozițional. Propoziția poate fi adevărată sau falsă. În situația în care $n = 0$ se spune că predicatul însuși este o propoziție.

Gramatical, predicatul instanțiat $CALCULATOR$ (IBM PC) reprezintă declarația "IMB PC este un calculator" care este fie adevărată, fie falsă. În exemplul dat, prima afirmație (i) este o propoziție declarativă ce conține cel puțin o variabilă, a doua (ii) nu este o propoziție, iar ultima (iii) devine o propoziție cînd fiecare din variabile sînt instanțiate. Terminologic setul obiectelor ce sînt variabile în propoziție poartă denumirea de "universul discursului".

În structura calculului cu predicate sînt incluși și cuantificatori. Se scrie $\forall x(FUNCTIONEAZĂ(x))$ pentru a indica faptul că orice calculator x este în funcțiune. Simbolul \forall numit cuantificator universal înlocuiește expresia

"pentru toate" sau "pentru oricare". Similar, cuantificatorul existențial indică faptul că există cel puțin un x pentru care afirmația este adevărată. Cu aceste notații regula de mai sus poate fi transcrisă prin:

$$\forall x[\text{CALCULATOR}(x) \rightarrow \text{FUNCTIONEAZĂ}(x)]$$

CALCULATOR (IMB PC)

Deci, FUNCTIONEAZĂ (IMB PC) și poate fi citită "Pentru orice x care este calculator, x este în funcțiune". Schematic structura regulii poate fi ilustrată ca în fig. 4.2.

În general expresia

$$\forall x[A(x) \rightarrow S(x)]$$

indică faptul că setul tuturor elementelor marcate prin $A(x)$ adevărate fac ca $S(x)$ să devină adevărat. În concluzie, pentru orice instanță particulară x pentru care $T(A(x)) = 1$ sigur $T(S(x)) = 1$. Cu acestea se poate observa că un predicat se poate identifica cu o relație al doilea termen semnificând un subset al setului universal de obiecte. Presupunând ca setul u reprezintă universul discursului pentru predicatul $P(\cdot)$, și $P(x)$ este interpretat gramatical ca " x este un predicat", atunci pentru orice x din u , $P(x)$ este fie adevărat fie fals; dacă x aparține clasei de obiecte pentru care declarația " x este un predicat" este adevărată atunci

$$T(P(x)) = 1$$

pe când dacă x aparține clasei de obiecte pentru care declarația " x este un predicat" este falsă, atunci:

$$T(P(x)) = 0.$$

În consecință dacă se va nota cu P_0 clasa formată, atunci

$$T(P(x)) = 1$$

dacă și numai dacă $x \in P_0$.

În plus dacă $Q(\cdot)$ este un alt predicat, cu Q_0 indicând un subset al elementelor $x \in u$ pentru care $Q(x)$ este adevărat, atunci

$$T(\forall x[P(x) \rightarrow Q(x)]) = 1$$

dacă și numai dacă P_0 este un subset al lui Q_0 ($P_0 \subset Q_0$).

Exemplul 1: Se consideră setul universal Y constând din patru entități;

1. Un automobil în funcțiune, notat DAC.
2. Un element ce nu este în lucru, notat NOT.
3. Calculatorul IBM PC.
4. Alt calculator, numit BIT.

Se obține deci că universul problemei este $Y = \{\text{DAC}, \text{NOT}, \text{IBM PC}, \text{BIT}\}$. Se va ilustra mai jos modul în care predicatele $\text{CALCULATOR}()$ și $\text{FUNCTIONEAZĂ}()$ corespund relațiilor notate CALCULATOR și FUNCTIONEAZĂ . Dacă se presupune ca afirmația "toate calculatoarele funcționează" este adevărată, adică:

$$T(\forall x[\text{CALCULATOR}(x) \rightarrow \text{FUNCTIONEAZĂ}(x)]) = 1$$

Atunci pentru,

$$\text{CALCULATOR} = \{\text{IBM PC}, \text{BIT}\}$$

$FUNCTIONEAZ\grave{A} = \{DAC, IBM\ PC, BIT\}$

se obține,

$$T(FUNCTIONEAZ\grave{A} (IBM\ PC)) = 1$$

$$T(FUNCTIONEAZ\grave{A} (BIT)) = 1$$

$$T(FUNCTIONEAZ\grave{A} (DAC)) = 1$$

și

$$T(FUNCTIONEAZ\grave{A} (NOT)) = 0$$

De aceea se obține:

$$T(\forall x[FUNCTIONEAZ\grave{A} (x)]) = 0$$

$$T(\exists x[FUNCTIONEAZ\grave{A} (x)]) = 1.$$

Nu este necesar ca predicatele să fie unare și chiar mai mult de un cuantificator poate fi utilizat.

Este obișnuită reprezentarea faptelor în bazele de cunoștințe ca fișiere extinse. Un fișier de acest tip, utilizat de regulă în bazele de date relaționale, apare ca o matrice pentru afișarea n-uplurilor în relațiile n-are; cu alte cuvinte este un subset al produsului $E_1 \times E_2 \times \dots \times E_n$, în care E_i este un set ce indică al i-lea obiect. Fiecare rînd al fișierului conține un n-uplu, rîndurile fiind elemente ale relației. Ca exemplu, relația $FUNCTIONEAZ\grave{A}$ din exemplul anterior poate fi exprimată:

IBM PC
DAC
BIT

sau în altă ordine, întrucît ordinea este irelevantă. O consecință este aceea ca informația este reprezentată în calculator în forma unor fișiere extinse. Aceasta nu necesită reprezentarea directă, ci poate fi dată în alte fișiere în conjuncție cu regulile ce implică conective ale calculului propozițional, predicatelor și cuantificatorilor.

Exemplu 2: Se consideră imaginea din figura 4.3, în care se observă mai multe obiecte: o casă, un arbore, o mașină. Cinci persoane pot fi observate în această imagine: Ion și Mihai în casă la fereastră, Ana în fața ușii casei, Mircea în automobil, Vasile ascuns în spatele pomului. Întreaga scenă poate fi bine descrisă prin introducerea unor predicate binare STÎNGA (\cdot, \cdot) și JOS (\cdot, \cdot). Dacă obiectul x este în stînga obiectului y atunci

$$T(STÎNGA(x, y)) = 1$$

Dacă obiectul w este mai jos decît obiectul u , atunci

$$T(JOS(w, u)) = 1$$

Se obține deci

$$T(STÎNGA(casă, automobil)) = 1$$

$$T(STÎNGA(casă, arbore)) = 1$$

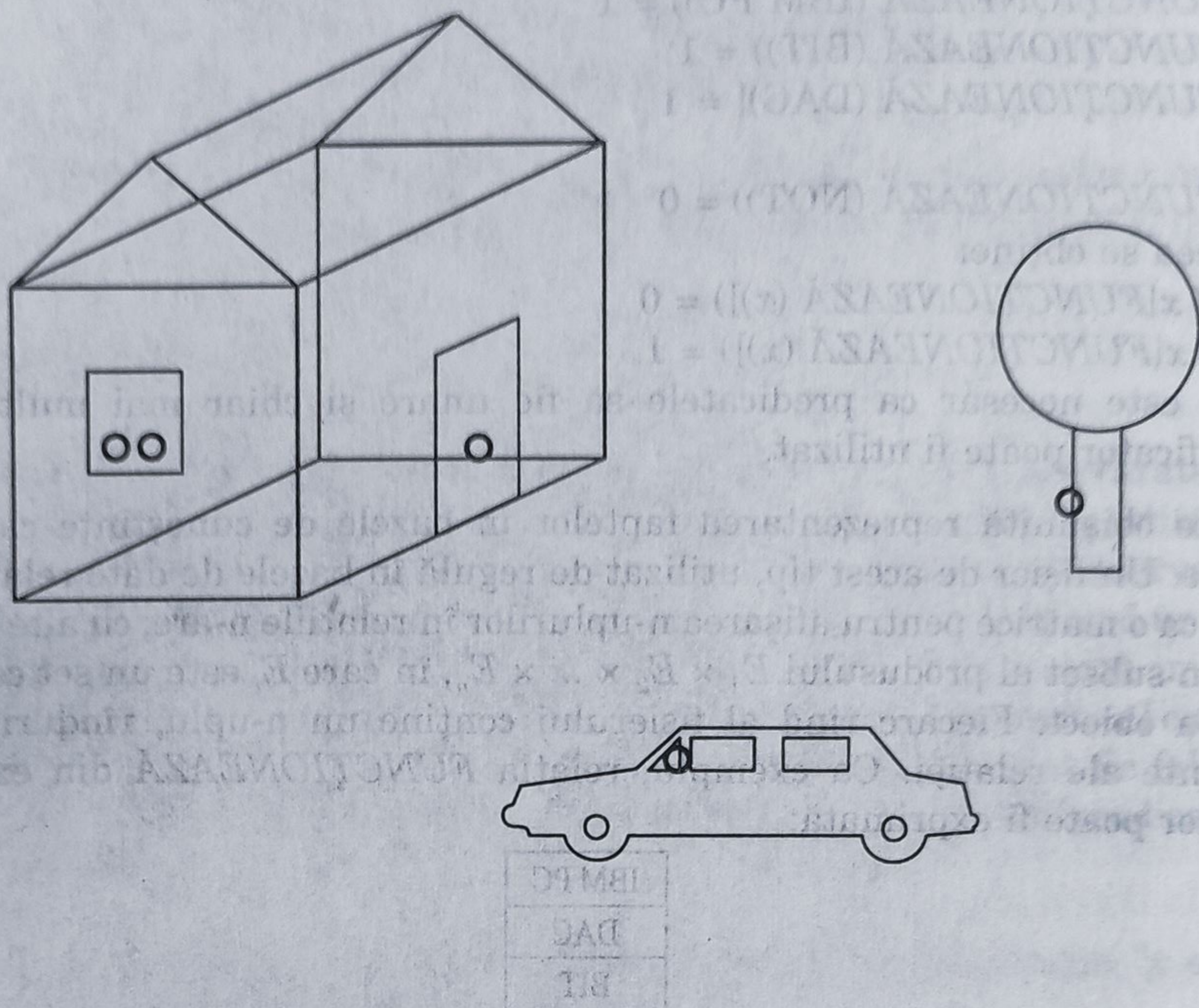
$$T(STÎNGA(automobil, arbore)) = 1$$

$$T(JOS(automobil, arbore)) = 1$$

$$T(JOS(automobil, casă)) = 1$$

$$T(JOS(arbore, casă)) = 1$$

▼ Fig. 4.3. Scena pentru exemplul 2



Informațiile precedente pot fi văzute în termeni ai relațiilor sau în termeni ai fișierelor extinse în memoria calculatorului, prezentându-se numai informațiile pentru care predicatele au valoarea "true" astfel:

STINGA

casă	automobil
automobil	arbore
casă	arbore

JOS

automobil	arbore
automobil	casă
arbore	casă

În maniera similară se pot defini predicatele binare $IN(,)$, $ASCUNS(,)$, ce duc la relațiile:

IN

Ion	casă
Mihai	casă
Mircea	automobil

ASCUNS

Vasile	arbore
--------	--------

Se poate ușor demonstra modul în care calculatorul determină când secvența cuantificată "Toate persoanele sînt în casă"

$\forall x[IN(x, casa)]$

este adevărată sau falsă. Prin fixarea variabilei $y = casă$, $IN(x, casă)$ este un predicat unar obținut din $IN(x, y)$ descris prin fișierul de mai jos.

IN (x, casă)

Ion
Mihai

Instantierea variabilei x pentru toate persoanele produce propoziția p definită mai jos în fig. 4.4.

▼ Fig. 4.4. Propoziția p

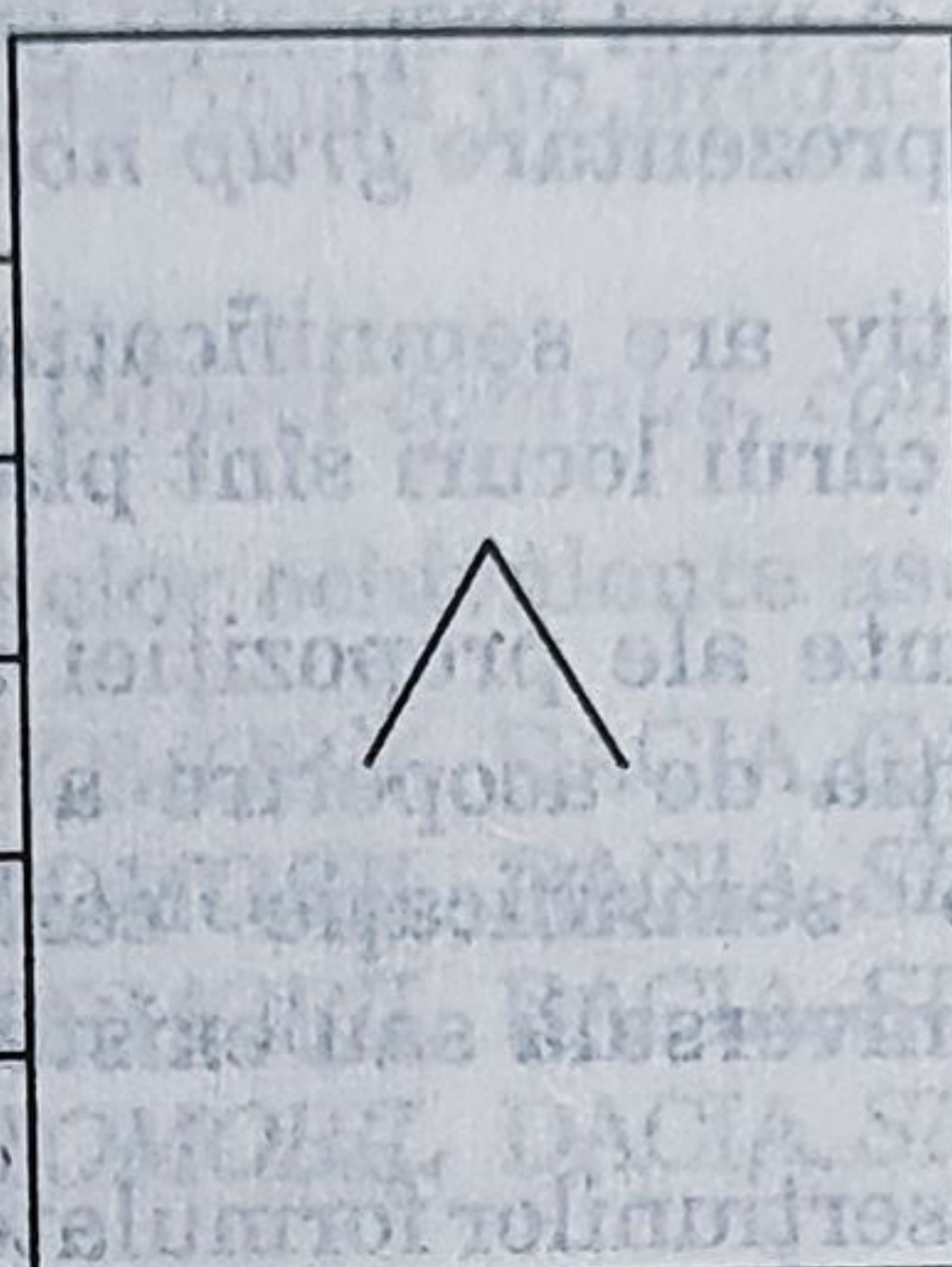
IN (Ion, casă)

IN (Mihai, casă)

IN (Mircea, casă)

IN (Ana, casă)

IN (Vasile, casă)



P

Valoarea de adevăr pentru p este 0 întrucât cele cinci conjuncții nu au toate valoarea adevărată. De exemplu, $T(\text{IN}(\text{Mircea, casă})) = 0$, întrucât înregistrarea (Mircea, casa) nu apare în fișierul extins IN. Deci, se poate afirma că $T[\forall(x[\text{IN}(x, \text{casă})])] = 0$. Similar valoarea de adevăr a unei cereri complexe poate fi realizată prin

$$T(\exists x \exists y \exists z \exists w [\text{IN}(x, y) \vee \text{JOS}(y, z) \vee \text{JOS}(w, z)]) = 1$$

ce este determinată ca disjuncția propozițiilor corespunzătoare cu variabilele instanțiate și determinarea faptului că cel puțin una este adevărată.

Se consideră următoarea descriere:

"Automobilul DACIA este compus din caroserie, sistem de propulsie, sistem de comandă și sistem de rulare. Automobilul se deplasează dacă are viteza relativă diferită de zero față de obiectele fixe. Motorul funcționează dacă are combustibil în rezervor și este închis contactul de pornire".

Cele trei fraze pot fi descompuse în mai multe aserțiuni astfel:

- I - A1 - Automobilul DACIA este compus din caroserie, sistem de propulsie, sistem de comandă și sistem de rulare.
- II - A2 - automobilul se deplasează.
- A3 - viteza relativă față de obiectele fixe este diferită de zero.
- III - A4 - motorul funcționează.
- A5 - are combustibil în rezervor
- A6 - este închis contactul de pornire.

Presupunînd că nu se cunosc alte fapte despre autoturism se poate transpune această cunoaștere în următoarele formule ale calculului propozițional.

I $A1$

II $A3 \rightarrow A2$

III $(A5 \wedge A6) \rightarrow A4$

Se observă că această reprezentare nu surprinde detalii ale descrierii inițiale întrucît acestea nu sînt propoziții ci componente ale acestora. Propozițiile sînt presupuse a fi bine alcătuite după reguli ce aparțin limbajului natural nu calculului propozițional. Pentru aceasta rafinarea presupune efectuarea unei analize ce se bazează pe reprezentarea predicativă în care structura gramaticală a unei propoziții este de forma *subiect-predicat* sau în forma apropiată de reprezentare *grup nominal, grup predicativ*.

- ♦ grupul predicativ are semnificația abstractizată printr-un simbol de predikat în ale cărui locuri sînt plasate obiecte ale grupului nominal;
- ♦ unele componente ale propoziției au rolul de a modifica valoarea de adevăr în funcția de acoperire a domeniului în care variabilele iau valori dînd o semnificație cantitativă ce este surprinsă prin cuantificarea universală sau existențială a variabilelor.

Pentru descrierea aserțiunilor formulate în specificarea piesei de cunoaștere automobilul DACIA, se obține pentru aserțiunea $A1$ o reprezentare ce pornește de la stabilirea unui predikat specific. Grupul predicativ al aserțiunii $A1$ determină un predikat ce va fi notat prin simbolul *ESTE_COMPUS* cu un număr de obiecte ale grupului nominal dat de numărul obiectelor din aserțiunea $A1$, adică "caroserie, sistem de propulsie, sistem de comandă, sistem de rulare" (5 obiecte în care primul este ocupat de obiectul compus, în cazul de față automobil).

♦ Aserțiunea $A1$

Astfel, în termeni de variabile formale predikatul ce generează acest tip de aserțiuni are forma generală:

ESTE_COMPUS (obiect-compus, componenta 1,... componenta 4)

în care, instanțierile de obiecte ce generează aserțiunea $A1$ fac predikatul adevărat. Se va obține pentru aserțiunea $A1$:

ESTE_COMPUS (AUTOMOBIL DACIA, CAROSERIE, SISTEM DE PROPULSIE, SISTEM DE COMANDĂ, SISTEM DE RULARE)

Această formulare, chiar dacă are calitatea condensării întregii cunoașteri despre automobil într-un singur predikat, are următoarele dezavantaje:

- în timpul raționamentelor nu se utilizează decît rareori toate componentele simultan, utilizîndu-se cel mai frecvent o singură componentă;

- schimbarea numărului de locuri ale predicatelor determină redefinirea predicatului, la rafinarea sa ulterioară fiind necesară mărirea arității sale dacă numărul componentelor crește. Rezolvarea acestor inconveniente poate fi făcută prin reformarea aserțiunii într-o formă logic echivalentă prin conjuncții de propoziții elementare al căror grup predicativ conduce la predicate binare.

Se obține astfel pentru aserțiunea A1:

A1 (caroseria este componentă a automobilului DACIA)

(sistemul de propulsie este componentă a automobilului DACIA)

(sistemul de comandă este componentă a automobilului DACIA)

(sistemul de rulare este componentă a automobilului DACIA).

Grupul predicativ nou obținut determină un predicat cu două locuri avînd structura:

ESTE_COMPONENT (obiect-compus, componenta)

Instantele corespunzătoare formelor echivalente aserțiunii A1 vor fi:

(1) *ESTE_COMPONENT (AUTOMOBIL DACIA, CAROSERIE)*

(2) *ESTE_COMPONENT (AUTOMOBIL DACIA, SISTEM DE PROPULSIE)*

(3) *ESTE_COMPONENT (AUTOMOBIL DACIA, SISTEM DE COMANDĂ)*

(4) *ESTE_COMPONENT (AUTOMOBIL DACIA, SISTEM DE RULARE)*

Aceasta este o posibilă reprezentare a aserțiunii A1 utilizînd calculul cu predicate de ordinul întîi. În această situație rafinarea presupune adăugarea de noi instanțe ale aceleiași expresii predicative cum este exemplul:

ESTE_COMPONENT (AUTOMOBIL DACIA, SISTEM DE FRÎNARE)

- ♦ Aserțiunea A2 - grupul predicativ este alcătuit din verbul "se deplasează", iar grupul nominal determină pentru predicatul respectiv un singur loc obținînd forma generală:

SE DEPLASEAZĂ (nume_obiect)

obținînd instanța: *SE DEPLASEAZĂ (AUTOMOBIL DACIA)*

- ♦ Aserțiunea A3 - predicatul definit are semnificația "este_diferită" ce indică două locuri ce corespund la două variabile formale (x, y). Se poate folosi în acest caz predicatul

ESTE_DIFERIT (x, y)

obținînd instanța: *ESTE_DIFERIT (VITEZA_AUTOMOBIL, 0)*

- ♦ Aserțiunea A4 - predicatul este "funcționează (x)" iar obiectul ce face adevărat acest predicat și generează o propoziție echivalentă cu aserțiunea A4 este motorul. Obținem în acest caz reprezentarea:

FUNCȚIONEAZĂ (nume_obiect)

cu instanța: *FUNCȚIONEAZĂ (MOTOR)*

- ♦ Aserțiunea A5 - folosește un predicat cu două locuri de forma generală:

ARE (recipient, conținut)

obținînd instanța: *ARE* (REZERVOR, COMBUSTIBIL)

- ♦ Aserțiunea A6 - utilizează un predicat cu un singur loc de forma:

ÎNCHIS (contact_electric)

obținînd instanța: *ÎNCHIS* (CONTACT_PORNIRE)

În concluzie expresiile determinate prin reprezentarea propozițională în limbajul calculului cu predicate de ordinul întâi și făcînd convenția să se omită conectiva logică conjuncție (\wedge) între propozițiile elementare, respectiv păstrarea conectivei logice disjuncție (\vee) în interiorul formulelor se obține:

(1) *ESTE_COMPONENT* (AUTOMOBIL D1300, CAROSERIE)

ESTE_COMPONENT (AUTOMOBIL D1300, SISTEM DE PROPULSIE)

ESTE_COMPONENT (AUTOMOBIL D1300, SISTEM DE COMANDĂ)

ESTE_COMPONENT (AUTOMOBIL D1300, SISTEM DE RULARE)

(2) *DIFERIT* (VITEZA_AUTOMOBIL, 0) \rightarrow *SE DEPLASEAZĂ* (AUTOMOBIL DACIA)

(3) (*ARE* (REZERVOR, COMBUSTIBIL) *ÎNCHIS* (CONTACT_PORNIRE)) \rightarrow *FUNCȚIONEAZĂ* (MOTOR)

În acest exemplu de reprezentare predicatele exprimă obiecte bine precizate pentru obiectul unic DACIA. În situația în care se reprezintă conceptul generic automobil ce se instanțează în obiecte individuale ca D1300, FIAT,... etc. se poate spune că a fi automobil este o caracteristică a obiectului respectiv ce se exprimă printr-un predicat, ca de exemplu *AUTOMOBIL* \leftarrow (x). Predicatul va fi adevărat pentru orice obiect individual ce este în realitate automobil. Cum numărul de automobile individuale este foarte mare, a scrie pentru fiecare că are caroserie, sistem de propulsie... etc. este neconvenabil, motiv pentru care se vor utiliza cuantificatori universali.

Astfel utilizarea cuantificatorilor universali în cazul tipic determinat mai sus conduce la:

$(\forall) x \text{ AUTOMOBIL } (x) \rightarrow \text{ARE } (x, \text{CAROSERIE})$

$(\forall) x \text{ AUTOMOBIL } (x) \rightarrow \text{ARE } (x, \text{SISTEM DE PROPULSIE})$

$(\forall) x \text{ AUTOMOBIL } (x) \rightarrow \text{ARE } (x, \text{SISTEM DE COMANDĂ})$

$(\forall) x \text{ AUTOMOBIL } (x) \rightarrow \text{ARE } (x, \text{SISTEM DE RULARE})$

În acest caz se poate considera că toate obiectele fizice menționate sînt concepte generice nu individuale. Pentru aceasta se vor înlocui conceptele generice: *CAROSERIE*, *SISTEM DE PROPULSIE*, etc. cu variabile simbolice, iar caracteristica se specifică prin predicate cum s-a procedat la conceptul de automobil.

$\forall x \text{ CAROSERIE } (x) \rightarrow \text{ESTE ELEMENT } (x, C)$

în care, C este o mulțime de descrieri ale caroseriilor, diferite după formă și caracteristici ce se încadrează în conceptul generic *CAROSERIE*.

Predicatul *ESTE ELEMENT* semnifică apartenența obiectului atașat variabilei formale x la mulțimea C a caroseriilor. Problemele de apartenență la o mulțime pentru entități ce sînt termeni ai altor predicate pot fi rezolvate prin folosirea predicatelor dar conduc prin substituție la predicate de ordin superior. Aceste inconveniente pot fi rezolvate prin utilizarea ca termeni ai predicatelor a simbolurilor funcționale. Se poate scrie astfel:

$$CAROSERIE : X \rightarrow C$$

văzută ca o aplicație de la mulțimea X a automobilelor la mulțimea C a caroseriilor situație în care termenul $CAROSERIE(x)$ are ca valoare un simbol identificator de caroserie din mulțimea C a acestora. Se obține astfel:

$$\forall x \text{ AUTOMOBIL}(x) \vdash ARE(x, CAROSERIE(x))$$

ce este o expresie bine formată în calculul cu predicate de ordinul întâi. În situația cînd nu toate elementele unei mulțimi satisfac o formulă, însă există cel puțin un element care o satisface se utilizează cuantificatorul existențial al variabilei formale ce se atașează elementului.

Astfel pentru a reprezenta aserțiunea:

"Cineva merge la piață"

se poate folosi formula: $(\exists) x \text{ OM}(x) \wedge \text{MERGE}(x, \text{PIAȚĂ})$.

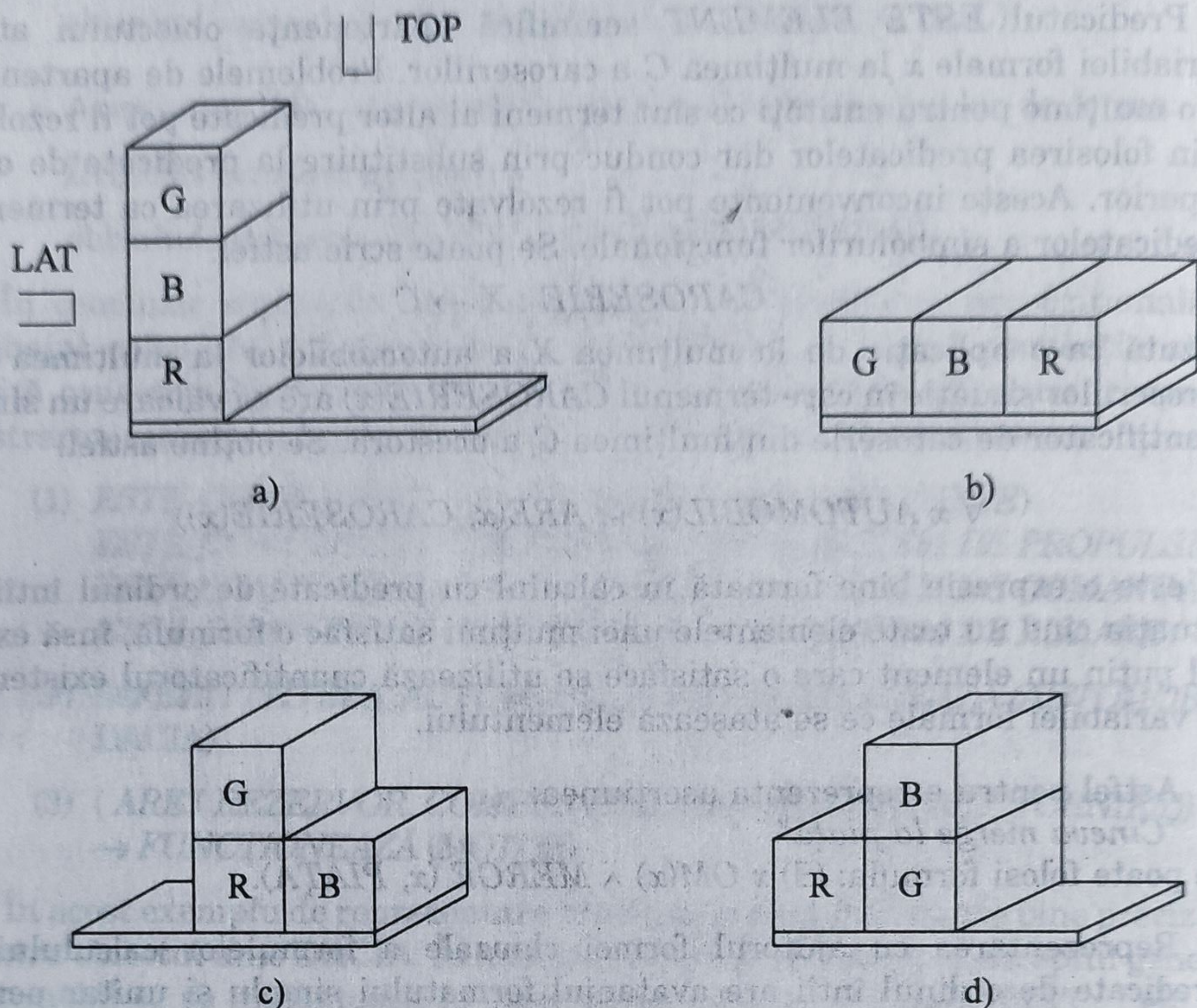
Reprezentarea cu ajutorul formei clauzale a formulelor calculului cu predicate de ordinul întâi are avantajul formatului simplu și unitar pentru toate categoriile constituente ale cunoașterii. Se poate cu ajutorul clauzei exprima într-o formulă compactă caracteristicile automobilului formulate mai sus.

$$\begin{aligned} \forall x \text{ ARE}(x, \text{CAROSERIE}) \wedge \\ \text{ARE}(x, \text{SISTEM DE PROPULSIE}) \wedge \\ \text{ARE}(x, \text{SISTEM DE COMANDĂ}) \wedge \\ \text{ARE}(x, \text{SISTEM DE RULARE}) \rightarrow \text{AUTOMOBIL}(x) \end{aligned}$$

Un exemplu tipic de utilizare a calculului cu predicate este cazul controlului unui robot. Pentru a ilustra modul în care un robot înțelege informațiile percepute utilizînd calculul cu predicate, se consideră lumea exterioară ca fiind formată dintr-o masă pe care sînt dispuse trei blocuri diferit colorate. Problema care se pune este relativă la modul de rearanjare a celor trei blocuri știind că pot fi așezate toate cele trei blocuri pe masă sau pot fi așezate în stivă unul peste altul, sau orice combinație a celor două așezări.

Se presupune că robotul a fost programat ca posibilă configurație a blocurilor să fie cunoscută prin colectarea informațiilor referitoare la poziție de la doi senzori, informații ce specifică culoarea blocului din fața sa. Un senzor percepe informațiile referitoare la așezarea pe verticală, senzor care este așezat lateral și celălalt informația referitoare la așezarea pe orizontală,

▼ Fig. 4.5. Configurații posibile ale blocurilor



Imaginînd un sistem de coordonate cartezian, cele 9 poziții posibile ale blocurilor pot fi reprezentate printr-o serie de predicate ce descriu pozițiile relative într-o grilă ca cea din fig. 4.6.

Se vor introduce predicatele $LOC(z, i, j)$ ceea ce indică faptul că blocul de culoare z este poziționat la locația (i, j) a sistemului de coordonate. Întregul i indică locația pe axa x ce este determinată prin senzorul TOP, întregul j locația pe axa y determinată de senzorul LAT. Predicatul $TOP(z, i)$ va fi utilizat pentru reprezentarea ieșirii senzorului superior, în care i reprezintă poziția corespunzătoare pe axa x . Analog senzorul lateral indică poziția blocului de culoare z pe axa y . Pentru aceasta predicatul $LAT(z, j)$ a fost introdus.

Pentru configurația ilustrată în fig. 4.5a se pot scrie valorile de adevăr ale predicatelor ca mai jos.

$$T(LOC(g, 0, 2)) = 1$$

$$T(LOC(b, 0, 1)) = 1$$

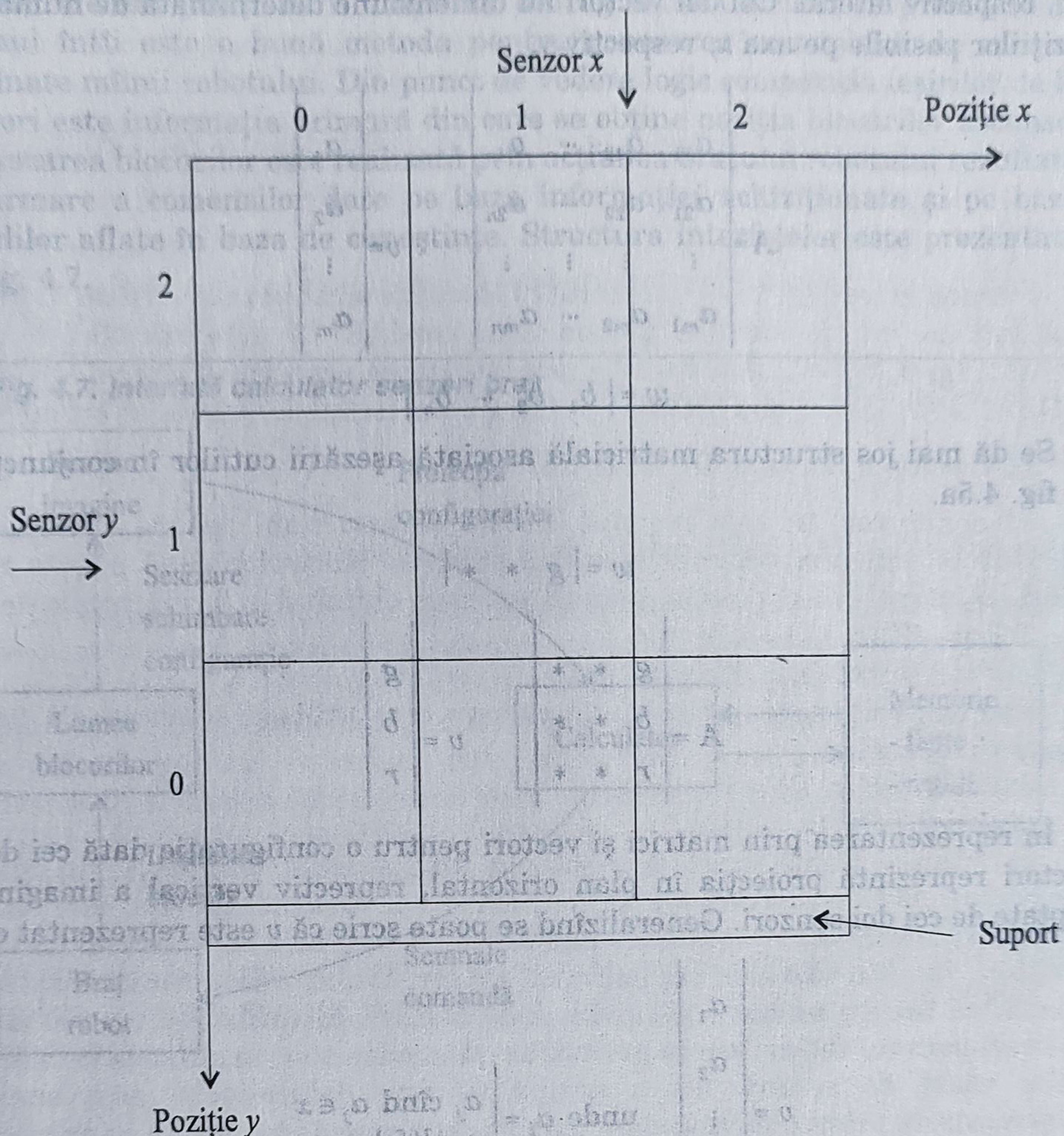
$$T(LOC(r, 0, 0)) = 1$$

$$T(LAT(g, 2)) = 1$$

$$T(LAT(b, 1)) = 1$$

$$T(LAT(r, 0)) = 1$$

▼ Fig. 4.6. Coordonate blocuri



$$T(\text{TOP}(g, 0)) = 1$$

Cu aceste informații pot fi scrise relațiile

$$\text{LOC} = ((g, 0, 2), (b, 0, 1), (r, 0, 0))$$

$$\text{LAT} = ((g, 2), (b, 1), (r, 0))$$

$$\text{TOP} = ((g, 0))$$

corespunzătoare localizării poziției pe axele x și y a celor trei blocuri.

Reprezentarea obiectelor poate fi specificată prin matrici sau vectori. O matrice de acest tip este de dimensiune $m \times n$ în care elementele aparțin setului de elemente X , absența unui element într-o poziție fiind marcată prin *.

Elementul a_{11} indică localizarea primului rînd și a primei coloane în sistemul de coordonate. Vectorii linie și coloană reprezintă poziția indicată de senzorii top, respectiv lateral. Cei doi vectori au dimensiune determinată de numărul pozițiilor posibile pe axa x , respectiv y .

$$A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix} \quad v = \begin{vmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{vmatrix}$$

$$w = \begin{vmatrix} b_1 & b_2 & \dots & b_n \end{vmatrix}$$

Se dă mai jos structura matricială asociată așezării cutiilor în conjuncție cu fig. 4.5a.

$$w = \begin{vmatrix} g & * & * \end{vmatrix}$$

$$A = \begin{vmatrix} g & * & * \\ b & * & * \\ r & * & * \end{vmatrix} \quad v = \begin{vmatrix} g \\ b \\ r \end{vmatrix}$$

În reprezentarea prin matrici și vectori pentru o configurație dată cei doi vectori reprezintă proiecția în plan orizontal, respectiv vertical a imaginii captate de cei doi senzori. Generalizînd se poate scrie că v este reprezentat ca

$$v = \begin{vmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{vmatrix} \quad \text{unde } a_i = \begin{vmatrix} a_j \text{ cînd } a_j \in x \\ * \text{ altfel} \end{vmatrix}$$

Similar proiecția pe axa y determină vectorul

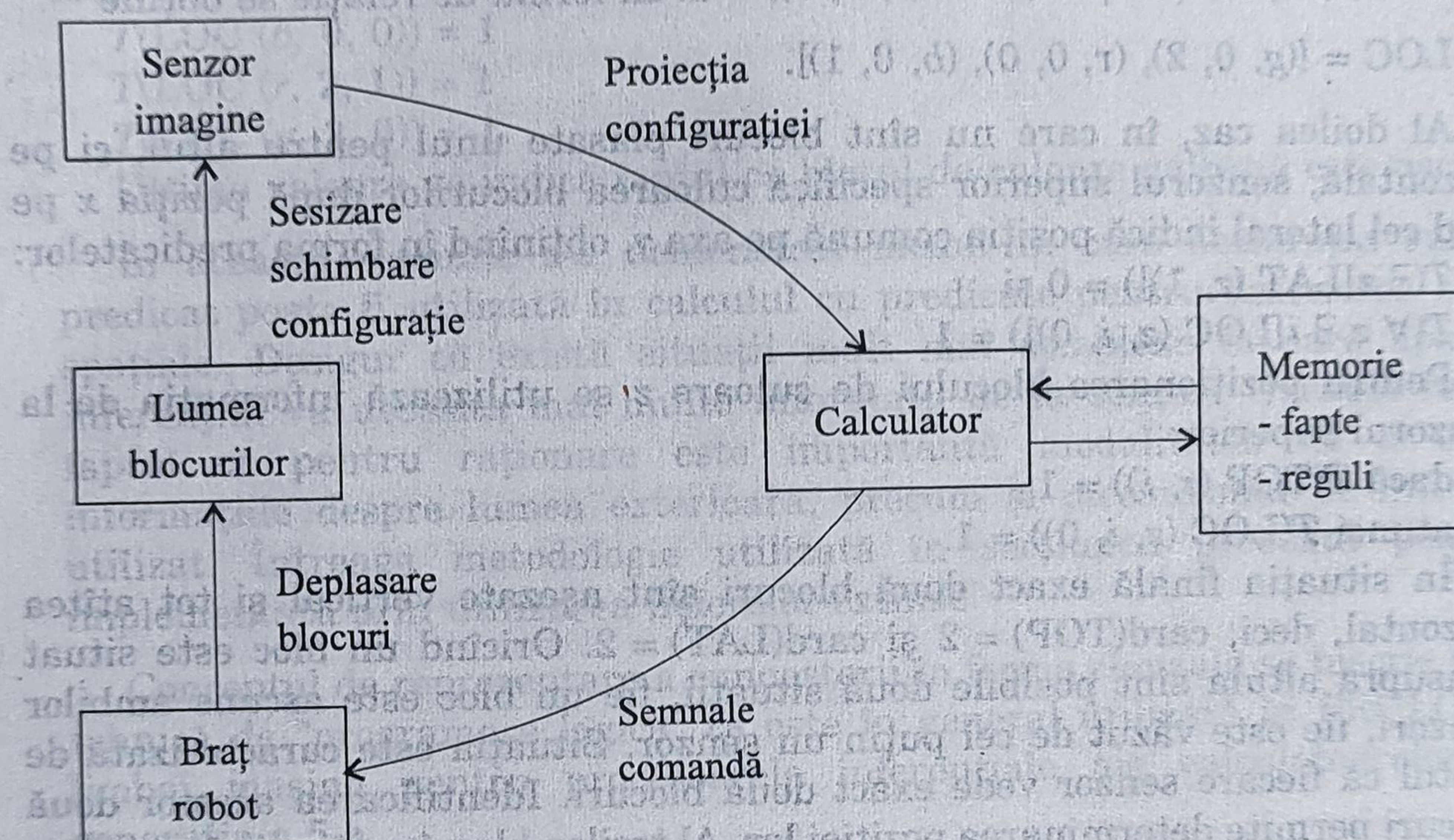
$$w = \begin{vmatrix} b_1 & b_2 & \dots & b_n \end{vmatrix}$$

$$\text{unde } b_i = \begin{vmatrix} b_j \text{ cînd } b_j \in x \\ * \text{ altfel} \end{vmatrix}$$

O problemă esențială este găsirea matricii A atunci cînd se cunosc vectorii proiecției $P_x(A) = v$ și $P_y(A) = w$. În general problema nu are soluție. Totuși, în multe cazuri pornind de la cunoștințele despre poziția blocurilor aferent informației obținute de la senzori se poate obține o unică matrice reprezentînd

imaginea blocurilor. În alte cazuri datorită faptului că un bloc este ascuns de altele nu este posibilă determinarea poziției exacte a acestora. Un astfel de caz este cel aferent poziționării blocurilor din fig. 4.5d. Calculul cu predicate de ordinul întâi este o bună metoda pentru generarea semnalelor de control destinate mîinii robotului. Din punct de vedere logic compoziția ieșirilor de la senzori este informația primară din care se obține poziția blocurilor ascunse. Reașezarea blocurilor este realizată prin acțiunea brațului robotului rezultată ca urmare a comenzilor date pe baza informației achiziționate și pe baza regulilor aflate în baza de cunoștințe. Structura interfețelor este prezentată în fig. 4.7.

▼ Fig. 4.7. Interfață calculator senzori braț



Brațul robotului reconfigurează poziția blocurilor, senzorii determină schimbarea configurației, iar calculatorul prin intermediul informației apriorice conținute în regulile logice determină poziția finală a noii configurații a blocurilor. Când configurația nu este cea dorită, semnale de control sînt transmise către brațul robotului pentru a determina acesta să schimbe poziția blocurilor către configurația dorită.

Se consideră cazul în care blocurile sînt așezate vertical unul peste altul. În această situație senzorul TOP va furniza locația comună x a tuturor blocurilor, senzorul lateral indică culoarea fiecărui bloc în conjuncție cu poziția lor pe axa y . În TOP apare numai un element, fapt pentru care $\text{card}(\text{TOP}) = 1$ și $\text{card}(\text{LAT}) = 3$ (prin $\text{card}()$ s-a notat numărul elementelor mulțimii diferite de *). Situația este în acest moment caracterizată prin

$$T(\exists z[\text{LAT}(z, 2)]) = 1.$$

În plus $T(\text{LOC}(z, i, 2)) = 1$, unde locația orizontală i a blocului din vârful stivei de culoarea z este determinată prin $T(\text{TOP}(z, i)) = 1$. Mai mult, alte două blocuri au aceeași poziție orizontală i și așezarea lor este detectată de senzorul lateral. Dacă $T(\text{LAT}(u, 1)) = 1$, atunci $T(\text{LOC}(u, i, 1)) = 1$, respectiv dacă $T(\text{LAT}(w, 0)) = 1$, atunci $T(\text{LOC}(w, i, 0)) = 1$.

În situația ilustrată în fig. 4.5a se obține pentru ieșirile senzorilor

$$\text{LAT} = [(g, 2), (b, 1), (r, 0)]$$

$$\text{TOP} = [(g, 0)]$$

Întrucît $T(\text{LAT}(g, 2)) = 1$, rezultă că blocurile sînt așezate vertical în prima coloană cu cel de culoare galben sus, condiție ce este specificată prin $T(\text{LOC}(g, 0, 2)) = 1$. Mai departe prin utilizarea senzorului lateral se obține $T(\text{LOC}(b, 0, 1)) = 1$ și $T(\text{LOC}(r, 0, 0)) = 1$. În formă de relație se obține

$$\text{LOC} = [(g, 0, 2), (r, 0, 0), (b, 0, 1)].$$

Al doilea caz, în care nu sînt blocuri plasate unul pentru altul, ci pe orizontală, senzorul superior specifică culoarea blocurilor după poziția x pe cînd cel lateral indică poziția comună pe axa y , obținînd în forma predicatelor:

$$T(\exists z[\text{LAT}(z, 1)]) = 0 \text{ și}$$

$$T(\forall z \exists i[\text{LOC}(z, i, 0)]) = 1.$$

Pentru poziționarea blocului de culoare z se utilizează informația de la senzorul superior:

$$\text{dacă } T(\text{TOP}(z, i)) = 1$$

$$\text{atunci } T(\text{LOC}(z, i, 0)) = 1.$$

În situația finală exact două blocuri sînt așezate vertical și tot atîtea orizontal, deci, $\text{card}(\text{TOP}) = 2$ și $\text{card}(\text{LAT}) = 2$. Oricînd un bloc este situat deasupra altuia sînt posibile două situații, fie un bloc este ascuns ambilor senzori, fie este văzut de cel puțin un senzor. Situația este caracterizată de faptul că fiecare senzor vede exact două blocuri. Identificarea acestor două blocuri permite determinarea poziției lor. Al treilea bloc, în situația în care nu este văzut de senzori are o poziție ce este determinată prin analiza informațiilor primite de la cei doi senzori. În termenii calculului cu predicate se obține:

$$T(\exists z \exists i[\text{TOP}(z, i) \wedge \text{LAT}(z, 0)]) = 1$$

și localizarea unui singur bloc de culoarea z este imediat găsită din informațiile ce provin de la cei doi senzori.

$$T(\text{LOC}(z, i, 0)) = 1.$$

Blocul de culoare u este deasupra și este observat de către cei doi senzori direct

$$T(\text{LOC}(u, j, 1)) = 1.$$

În final, întrucît $z \neq u$ și ambele sînt elemente ale lui $X = \{r, g, b\}$, blocul ascuns este elementul w , a cărui locație va fi specificată prin $T(\text{LOC}(w, j, 0)) = 1$.

Ultima situație corespunde cazului în care ambii senzori detectează două blocuri și numai unul dintre acestea este comun. Un bloc este deasupra altui

bloc și are o localizare obținută direct de la senzori. Un alt bloc văzut prin senzorul lateral are o localizare orizontală identică cu cea a blocului superior. În final, al treilea bloc trebuie să fie așezat pe masă și are o localizare orizontală dată de senzorul vertical. Utilizând predicatele, blocul superior de culoare z are nivelul 1 și $T(\text{TOP}(z, i)) = 1$. Deci, $T(\text{LOC}(z, i, 1)) = 1$. Blocul de jos de culoare u este văzut de senzorul lateral $T(\text{LAT}(u, 0)) = 1$ și $T(\text{LOC}(u, i, 0)) = 1$. Ultimul bloc are culoare $w \in X - \{z, u\}$ dată de senzorul superior și $T(\text{TOP}(w, j)) = 1$, implicând ca $T(\text{LOC}(w, j, 0)) = 1$.

Exemplu:

Considerînd că ieșirile celor doi senzori sînt date de relațiile:

$\text{TOP} = \{(b, 0), (r, 2)\}$ și

$\text{LAT} = \{(b, 0), (r, 1)\}$

$\text{card}(\text{TOP}) = 2$ și $\text{card}(\text{LAT}) = 2$, corespunde situației 3. Se va obține

$T(\text{LOC}(b, 0, 0)) = 1$

$T(\text{LOC}(r, 2, 1)) = 1$

$T(\text{LOC}(g, 2, 0)) = 1$.

Ultima valoare va indica faptul că blocul de culoare galbenă este ascuns.

În această secțiune s-a demonstrat modul în care declarația subiect-predicat poate fi utilizată în calculul cu predicate pentru descrierea scenei spațiale. Desigur că există situații mult mai complexe în care aparatul inferențial va necesita mai multe informații de la senzori. Trebuie reținut faptul că pentru raționare este importantă modalitatea de a obține informațiile despre lumea exterioară, precum și mecanismul de inferență utilizat. Întreaga metodologie utilizată în secțiunea prezentă poate fi implementată prin utilizarea fișierelor extinse.

Conceptul de reprezentare a cunoașterii în forma clauzală se înscrie într-o tehnică de "programare logică" ce este în general utilizată în PROLOG ca limbaj mașina pentru procesoarele inferențiale ale calculatoarelor din generația a 5-a.

4.2.2. Metode procedurale de reprezentarea cunoașterii

Reprezentarea procedurală nu este legată de cercetările în domeniul inteligenței artificiale. Programele de calcul cunoscute sînt piese de cunoaștere obținute prin reprezentarea procedurală a unor algoritmi. Reprezentarea cunoașterii asupra acestor entități utilizînd limbajul calculului cu predicate de ordinul întâi are dificultăți datorate caracterului declarativ al metodei, caracter ce nu corespunde cu natura procedurală a pieselor de cunoaștere. Reprezentarea declarativă implică reprezentări de multe ori artificiale pentru implementarea conceptului de control. Reprezentarea procedurală nu poate evita complet caracterul declarativ al pieselor de cunoaștere despre obiecte și fapte. Este motivul pentru care la ora actuală prin îmbinarea avantajelor celor două tipuri de reprezentări remarcabile. Metodele declarative sînt acele metode ce oferă facilități pentru specificarea aspectelor statice ale cunoașterii, pentru specificarea componentelor, proprietăților, evenimentelor, stărilor.

Metodele procedurale - se bazează pe aspectele dinamice ale cunoașterii asupra modului de folosire a pieselor de cunoaștere pentru efectuarea inferențelor, asupra determinării de noi fapte prin executarea unor noi operații asupra pieselor de cunoaștere. Metodele procedurale presupun utilizarea unor simboluri prin care se identifică proceduri ce sînt evaluate de procesoarele interpretative ale limbajelor de nivel înalt. Un exemplu de acest tip este limbajul LISP. Reprezentarea procedurală este realizată în termenii unor simboluri prin care cunoașterea este utilizată la rezolvarea problemelor. Astfel, fie piesa de cunoaștere oricît de complexă poate fi reprezentată procedural prin forma condensată următoare:

PENTRU_A_STABILI ORNITORINC (X);
DECLARĂ_PREDICATE (X);
DOVEDEȘTE MAMIFER (X);
DOVEDEȘTE DEPUNE_OUĂ (X);
DOVEDEȘTE ÎNOATĂ (X).

În această piesă de cunoaștere simbolurile *PENTRU_A_STABILI*, *DECLARĂ_PREDICATE* și *DOVEDEȘTE* sînt identificatori de proceduri prin care se urmărește:

- ▶ determinarea apartenenței unui obiect la o clasă;
- ▶ înscrierea proprietăților obiectului x în forma de predicate;
- ▶ determinarea valorii unui obiect al unei piese de cunoaștere.

Prima procedură le folosește pe celelalte drept componente în corpul reprezentării procedurale a cunoașterii. Ordinea de parcurgere este cea specificată efectul fiind că pentru a arăta că X este oricît de complex este necesar mai întîi să se descrie piesa de cunoaștere X prin intermediul predicatelor ce îi stabilește proprietățile, apoi se apelează procedura de determinare a valorii atributelor lui X . În schimb dacă se dau faptele că X este mamifer, depune ouă și că înoată, acestea pot servi la determinarea speciei de mamifer.

Desigur că reprezentarea procedurală este dependentă de tipul problemei ce se rezolvă cu piesele de cunoaștere reprezentate. În măsura în care cunoașterea devine tot mai complexă, piesele de cunoaștere devin tot mai numeroase. Metodele procedurale au ca principal avantaj ușurința aplicării regulilor existente, specifice domeniului de competență aplicativă. Cunoașterea astfel reprezentată este directă și conduce la performanțe de timp mult mai bune. Are însă dezavantajul major al flexibilității scăzute în utilizarea pieselor de cunoaștere pentru efectuarea altor inferențe decît cele specificate explicit prin procedurile aferente.

4.2.3. Reprezentarea cunoașterii prin rețele de producție

Acest mod de reprezentare a cunoașterii este una dintre cele mai utilizate în sisteme expert. Metoda se bazează pe separarea componentelor obișnuite ale calculului, în scopul manipulării ușoare în procesele la care sînt utilizate.

Cunoașterea în rețelele de producție este de natură procedurală și pot fi definite următoarele componente:

- ♦ *cunoaștere declarativă sau factuală* ce reprezintă piese de cunoaștere stocate sub forma unor structuri de date într-o colecție numită și *context*;
- ♦ *cunoaștere procedurală* reprezentată sub forma unei colecții de tip condiție-acțiune numite și *reguli de producție*, colecție ce formează *baza de reguli*;
- ♦ *cunoașterea strategică sau de control* formată din reguli ce privesc secvențele de acțiuni în procesul de rezolvare.

Sistemul construit în jurul regulilor de producție se bazează pe structura specifică compusă din cele două părți <partea de premisă> → <partea de acțiune>. Într-o altă exprimare un sistem de producție este compus dintr-o bază de date și un set de reguli. Condițiile unei reguli pot fi considerate ca o bază de date, ce returnează un indicator de succes sau eroare. Concluzia unei reguli este o acțiune ce manipulează date din baza de date, și în plus un control al sistemului determină secvența regulilor utilizate.

În descriere generală se poate menționa faptul că sistemele de producție sînt similare gramaticilor și apar chiar în definirea gramaticilor Chomsky. Astfel $G = (V, \Sigma, P, S)$ este o gramatică de structură a frazei, unde:

- V reprezintă un set finit de simboluri numit adesea și *alfabet total*;
- $\Sigma \subseteq V$ reprezintă un set finit de simbluri ale alfabetului numite *simboluri terminale* sau simplu *terminale*;
- P o submulțime a produsului cartezian $(V - \Sigma - s) * V$. Elementele lui P , deci perechile ordonate (u, w) , se mai scriu în mod obișnuit sub forma $u \rightarrow w$ și sînt numite *producții* sau reguli de rescriere;
- $\{s\}$ un simbol inițial sau simbol de start.

Se pornește de la ideea că un sistem de producții este compus dintr-o bază de date și un set de reguli. Condițiile unei reguli pot fi văzute încă similare cu o bază de date ce întoarce un indicator de succes sau eroare. Concluzia unei reguli este o acțiune ce manipulează baza de date și în plus un control va determina secvența regulilor utilizate. Baza de date conține cuvinte construite cu simboluri din V . În concluzie, un sistem de producție R poate fi considerat un dublet $R = (D, P)$, în care:

D este o bază de date

P set finit de reguli.

Baza de date este constituită dintr-un set de termeni, iar o regulă are forma generală IF c THEN t , în care condiția c este constituită din termeni, paranteze, conective \vee, \wedge, \neg , iar concluzia t este formată dintr-un singur termen. Termenii sînt constituiți din parametrii $x \in \mathcal{P}(R)$ și din valori $a \in \mathcal{V}(R)$. Se presupune că $\mathcal{P}(R)$ și $\mathcal{V}(R)$ sînt mulțimi finite și $\mathcal{P}(R) \cap \mathcal{V}(R) = \emptyset$.

Sintaxa $x = a$, respectiv $x \# a$ cu $x \in P(R)$ și $a \in V(R)$ va trebui interpretată că x este mărginit de a și, respectiv că x nu este mărginit de a . Un termen t este adevărat pentru baza de date D dacă și numai dacă $t \mid D$. Succesul sau eșecul unui termen arbitrar în condiția c este redus la succesul sau eșecul termenilor componenți.

Observații. 1. În general nu se permite o conjuncție de termeni la o concluzie. În acest sens o regulă de tipul

IF c THEN $t \ \& \ s$

se va scrie ca două reguli în forma

IF c THEN t

IF c THEN s

2. Definiția implică faptul că $\neg x = a$ nu este aceeași cu $x \neq a$ întrucât $x = a \in D$ nu este același cu $x \neq a \in D$. *Exemplu:* Fie R un sistem de producții cu:

$R = (\{x = a, y = b, z = a\}$

$\{ \text{IF } (x = a \wedge \neg y = c \wedge z \neq a) \text{ THEN } q \neq d \})$.

Întrucât la condiția regulii, baza de date returnează succes se va obține în baza de date $D' = \{x = a, y = b, z \neq a, q \neq d\}$.

Se poate afirma că structura generală a unei reguli de producție este următoarea:

$\langle \text{partea condiție} \rangle \rightarrow \langle \text{partea acțiune} \rangle$

avînd interpretarea,

IF $\langle \text{partea condiție} \rangle$ este îndeplinită,

THEN se execută $\langle \text{partea acțiune} \rangle$

sau în forma mai generală:

IF $\langle \text{partea condiție} \rangle$ este îndeplinită,

THEN se execută $\langle \text{partea acțiune}_1 \rangle$

ELSE se execută $\langle \text{partea acțiune}_2 \rangle$.

Dacă într-o regulă condiția este satisfăcută se spune că regula este selectată pentru declanșare sau regula este aplicabilă. Regulile aplicabile intră într-o mulțime a regulilor aplicabile, mulțime din care este selectată regula cu cea mai mare prioritate după criterii ce se vor prezenta. Mecanismul regulilor de producție este împrumutat din teoria limbajelor formale, este de natură procedurală și poate cel mai apropiat de modul de realizare a programelor clasice. Însăși structura regulii de tipul IF - THEN - ELSE este una din structurile de control întâlnite în toate limbajele de nivel înalt.

Mecanismul interpretativ al regulilor de producție conține următorii pași:

- ♦ selectarea tuturor regulilor ce conțin piese de cunoaștere ce satisfac partea de condiție numită și *corespondență*. Mulțimea acestor reguli formează *mulțimea candidată*, denumire ce are originea în faptul că elementele acestei mulțimi intră într-o competiție în urma căreia se decide care regulă este efectiv aplicată;
- ♦ rezolvarea conflictelor prin care din mulțimea regulilor aplicabile se elimină mai întîi regulile ce duc la aceleași rezultate, după care în conjuncție cu mecanismul de asertare a priorității pentru problema respectivă se selectează regula ce va fi efectiv aplicată;

- ♦ execuția părții *acțiune* a regulii cu cea mai mare prioritate, în situația în care sînt producții aplicabile;
- ♦ pornind de la contextul modificat în urma aplicării regulilor anterioare se reia ciclul începînd cu faza de corespondență, atîta timp cît ciclul produce acțiuni materializate prin modificarea contextului.

Oprirea mecanismului interpretativ poate avea loc dacă acțiunea unei producții specifică concret oprirea, sau se selectează o producție vidă.

Definiție: Un sistem de producții este un cvintuplu $SP = (K, P, \sigma, \varphi, p_s)$, în care,

K reprezintă contextul ca o mulțime de piese de cunoaștere factuală recunoscute de celelalte componente ale sistemului;

P baza de reguli ca mulțime finită de reguli de producție;

σ funcția succesori la îndeplinirea cu succes a condiției, funcție definită pe mulțimea P cu valori în mulțimea $P \cup \{\lambda\}$;

φ funcția succesori în caz de eșec a condiției ca o aplicație între aceleași mulțimi ca și funcția succesori la succes;

p_s regula de producție inițială de la care pornește procesul de selectare a regulilor.

S-a notat cu λ producția vidă.

Întrucît, partea condiție și partea acțiune sînt forme complexe ce conțin mai multe condiții și mai multe acțiuni forma cea mai generală a unei reguli de producție este:

$$p = (c_1 \wedge c_2 \wedge \dots \wedge c_i) \rightarrow (a_1; a_2; \dots a_p)$$

în care c_j , $1 \leq j \leq i$ sînt condițiile și a_k , $1 \leq k \leq p$ sînt acțiunile.

Pentru ca o producție p să fie selectată, ținînd cont de conectiva logică dintre condițiile c_j , este necesar ca în context să existe piese de cunoaștere ce satisfac toate aceste condiții. Dacă regula este selectată atunci acțiunile a_k , $1 \leq k \leq p$ sînt executate.

Controlul sistemelor de producții este specificat complet de cele trei componente ale sale σ , φ și p_s și deci se pot furniza principalele strategii de control. O metodă des întîlnită este cea prin care se concepe o *strategie exhaustivă* ce selectează toate regulile ce se potrivesc condiției date de context. O astfel de strategie este indicată mai jos.

Fie contextul $K = \{k_1, k_2, \dots, k_n\}$
și producțiile $P = \{p_1, p_2, \dots, p_m\}$

$$\begin{cases} \sigma(p_i) = \varphi(p_i) = p_{i+1}, \text{ pentru } 1 \leq i \leq m-1 \\ \sigma(p_n) = \varphi(p_n) = p^n \end{cases}$$

dacă, mulțimea candidată este diferită de mulțimea vidă și funcția de rezolvare a conflictelor de prioritate $c(MC)$ dă rezultatul p .

$$\sigma(p_n) = \varphi(p_n) = \lambda$$

dacă, mulțimea candidată este vidă sau funcția de rezolvare a conflictelor oferă rezultatul STOP.

$$p_s = p_1$$

Altă metodă de strategie de control este inspirată din algoritmi normali Markov:

$$K = \{k_1, k_2, \dots, k_n\}$$

$$P = \{p_1, p_2, \dots, p_m\}$$

$$\sigma(p_i) = p_1 \text{ pentru } 1 \leq i \leq n$$

$$\phi(p_i) = p_{i+1} \text{ pentru } 1 \leq i \leq n-1$$

$$\phi(p_n) = \lambda$$

$$p_s = p_1$$

Conform acestei strategii sistemul cercetează condițiile din context începând cu prima regulă, iar la întâlnirea primei reguli aplicabile o aplică după care comută controlul de regulă p_1 cu noul context. În situația de succes controlul este comutat la regula p_{i+1} realizând astfel o explorare succesivă. În condițiile în care nici o regulă nu este aplicabilă se selectează conform funcției succesor la eșec producția vidă.

O metodă des întâlnită este și cea utilizată în sistemul expert EXSYS, prin care se alege din mulțimea candidată producția cu cel mai mare câștig informațional. În mediul de dezvoltare EXSYS din mulțimea regulilor aplicabile, adică din cele care fac parte din mulțimea conflictuală, se va declanșa efectiv acea regulă care are în partea premisă mai multe condiții și în partea acțiune mai multe acțiuni posibile. Logica de mai sus se bazează pe faptul că o astfel de regulă conține mai multă informație, fapt ce va determina atingerea mai repede a obiectivului.

4.2.3.1. Analiza condiției

Funcție de forma condiției se va explora contextul pentru a găsi piese de cunoaștere ce corespund clauzelor constitutive ale condiției. O regulă are o exprimare narativă cum este de exemplu următoarea:

IF: domeniul de utilizare al calculatorului este cercetare științifică, sau proiectare asistată

(and) volumul programelor dezvoltate este foarte mare

(and) costul nu reprezintă un factor major

THEN: coprocesor matematic este necesar

(and) frecvența ceasului este mare

(and) ploter este necesar

(and) display SVGA.

Forma în care o regulă este introdusă în baza de reguli depinde de modul de realizare a interpretorului de reguli. Dacă se folosesc liste înlănțuite cum sînt de exemplu listele LISP condiția ce determină contextul se exprimă:

(Context (domeniul de utilizare al calculatorului este cercetare științifică, sau proiectare asistată) (volumul programelor dezvoltate este foarte mare) (costul nu reprezintă un factor major))).

Exprimarea regulii sub forma unei liste LISP ca în exemplul de mai sus duce la o structură de tipul următor:

REGULA nr. 5

(Context (domeniul de utilizare al calculatorului este cercetare științifică, sau proiectare asistată) (volumul programelor dezvoltate este foarte mare) (costul nu reprezintă un factor major))

(Acțiune (coprocesor matematic este necesar) (frecvența ceasului este mare) (ploter este necesar) (display SVGA))

Într-o reprezentare de acest tip se pot folosi chiar funcțiile LISP pentru corespondența cu partea de condiție cât și pentru realizarea operațiilor specifice acțiunii. Acest mod este inefficient din punctul de vedere al memoriei ocupate însă are avantajul simplității prin faptul că se utilizează facilități ale unui limbaj cunoscut. O astfel de reprezentare a cunoașterii a fost folosită în sistemul expert MYCIN.

Sistemele expert actuale înglobează mecanisme de evaluare a condiției mult mai evaluate fără să se restrângă la operația simplă de corespondență între condiție și context. Expertul, care este sursa cunoașterii poate prevedea orice evaluare de funcție, funcție care să reprezinte cel mai bine criteriul de acceptare a regulii de producție, însă reprezentarea nu mai este unică și modulară. Mecanismele de corespondență prezentate pînă în prezent au o dificultate majoră legată de tratarea cunoștințelor incomplete prin care se modelează enunțurile problemelor, fapt pentru care apare ca necesară introducerea variabilelor. Utilizarea variabilelor face posibil ca în sistemele de producții să se folosească reguli ce operează nu numai cu piese individuale, ci și cu concepte. Legarea variabilelor facilitează construcția de instanțe diferite pentru aceleași reguli. În principal prin legarea variabilelor se rezolvă moștenirea proprietăților paternurilor din condiție.

4.2.3.2. Metode de rezolvare a conflictelor

Întrucît nu este cunoscut rezultatul punerii în corespondență a condiției regulii cu contextul, mai multe reguli sînt selectate în mulțimea candidată și sistemul va trebui să decidă care dintre acestea vor trebui declanșate. Se poate ajunge la o astfel de situație datorită conflictelor între instanțele aceleiași reguli cât și datorita conflictelor între reguli diferite ce candidează la soluție ca urmare a realizării cu succes a corespondenței. Rezolvarea acestor conflicte este făcută pe baza informației ce provine din context, din baza de reguli sau surse combinate.

Dacă informația provine din context poate fi indicată o anumită importanță a pieselor de cunoaștere materializată printr-o pondere ce este o dată de intrare în mecanismul de evaluare a priorității. Dacă se notează cu p_i ponderea regulii i din baza de reguli următoarele criterii sînt mai des utilizate:

- ♦ prioritatea dată de cea mai importantă piesă de cunoaștere din regulă. Pentru alegerea regulii r , $p_r = \max p_i$ în care $1 \leq i \leq n$;

- ♦ prioritatea determinată de media aritmetică a ponderilor de importanță ce apar în regula

$$p_r = \frac{1}{n} \sum_{i=1}^n p_i;$$

- ♦ ordinea inversă dispersiei ponderilor de importanță a pieselor de cunoaștere, ce se calculează prin

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (p_i - p_r)^2$$

în care, p_r este media aritmetică calculată mai sus.

Contextul mai poate furniza informații ce nu depind de importanța pieselor de cunoaștere ci doar de utilitatea și activarea acestora. Informațiile de acest tip sînt înregistrate și actualizate de interpretorul sistemului de producție și se referă la:

- ♦ vechimea piesei de cunoaștere apreciată după data calendaristică a înscrierii acesteia în context;
- ♦ numărul de participări ale piesei de cunoaștere la regulile efectiv declanșate, începînd din momentul înscrierii piesei de cunoaștere;
- ♦ numărul de neutilizări, adică declanșări de producții ce nu folosesc piesa de cunoaștere considerată. Desigur că prin strategiile de rezolvare a conflictelor se apreciază mai mult piesele de cunoaștere mai recente și cu mai mare număr de utilizări în ultima perioadă.

O sursă de informații importantă în ceea ce privește rezolvarea conflictelor de prioritate este constituită pe bază de reguli. În acest sens se are în vedere posibilitatea ca la scrierea regulii de producție să se cunoască informații privind modul de determinare a succesului imediat. Nu este recomandată chemarea explicită întrucît afectează anatomia sistemului de reguli. Nici o altă metodă nu asigură o eficiență egală cu cea a specificării succesului. Se încearcă prin diverse metode să se elimine dezavantajul pierderii autonomiei, păstrîndu-se avantajul înlocuirii procesului de căutare printr-o specificație directă. O metodă des întîlnită este cea care folosește reguli de prestabilire a producțiilor succesoare avînd forma:

(CONFLICT - P)

(CONDȚIE (P))

(ACȚIUNE (SUCCESOR (P_1, P_2, \dots, P_r)))

unde P este o producție al cărui succesori se caută, iar (P_1, P_2, \dots, P_r) este șirul ordonat după prestație al candidaților la producția succesoare.

O metodă eficientă pentru determinarea succesului este cea prin care se specifică în regulă obiectivul ce trebuie să-l rezolve regula următoare, eliminînd în acest fel neajunsurile privind chemarea explicită a succesului.

Interpretorul sistemului de producție trebuie să-și memoreze stările determinate prin funcționarea sa, pentru dirijarea procesului interpretor cât și pentru acțiunea în caz de eroare.

Stările determinante ale interpretorului sînt similare cu cele folosite în cazul procesoarelor și anume:

- ♦ *stare veche* - se înscrie producția efectiv declanșată înaintea noului ciclu interpretativ, piesele de cunoaștere implicate în forma veche a contextului ce au suferit modificări;
- ♦ *starea curentă* - în care este înscrisă faza din ciclu, producția ce se evaluează și piesele de cunoaștere implicate, lista producțiilor implicate în fazele anterioare stării curente, starea acțiunii de rezolvare a conflictelor de prioritate, starea acțiunii în curs de desfășurare;
- ♦ *starea nouă* - este înscrisă producția de la care se reia ciclul următor.

Informațiile de stare pot fi folosite pentru rezolvarea conflictelor de prioritate în unul din modurile:

- ♦ cercetarea stării vechi spre a vedea dacă noua regulă prioritară este distinctă față de ultima regulă declanșată efectiv;
- ♦ întrucît strategia precedentă nu elimină în totalitate repetările, se constituie un istoric al declanșărilor efective ce este folosit în procesul de rezolvare a conflictelor de prioritate pentru faza de minimizare a mulțimii conflictuale. Complexitatea informației de stare și a mecanismului de comutație a acesteia, în vedere asigurării unei desfășurări controlate a ciclului de interpretare a descurajat mulți autori. Multe din interpretoarele de pînă acum sînt simple fără control de stare care să permită în orice moment rezolvarea condițiilor excepționale ce pot apare ca urmare a incompletitudinii contextului, a considerării cererilor urgente, a optimizării folosirii resurselor, a ridicării gradului de paralelism la efectuarea proceselor.

Rezolvarea conflictelor de prioritate se poate baza și pe faptul că structura regulilor de producție selectate este purtătoare de informație utilă pentru acest scop. O regulă P_P este un caz particular al unei reguli P_G dacă:

- ♦ P_P și P_G sînt identice;
 - ♦ toate clauzele condiționale ale regulii generale P_G se regăsesc printre clauzele condiționale ale regulii P_P ;
 - ♦ pentru fiecare clauză condițională din P_G ce specifică piese de cunoaștere în context, există în P_P o clauză care conține un subset de piese de cunoaștere, printre care și aceea specificată în clauza din P_G .
- În acest caz se preferă P_P pentru a satisface în întregime restricțiile specificate de regula P_G .

4.2.3.3. Transmiterea acțiunii

Este caracteristică pentru sistemele de producție comunicarea prin intermediul contextului între reguli. Metoda este folosită în programarea clasică, avînd ca principiu prevederea unui depozit de informație la dispoziția mai multor taskuri. Această zonă se mai numește și zonă de comunicație sau cutie poștală. Metoda întâlnită în literatura numită și "comunicație prin date", abordează programul nu ca pe o secvență prestabilită de operații, ci realizează înlănțuirea operațiilor ca urmare a interpretării conținutului cutiei poștale după reguli ce permit stabilirea relației următoare. Aceste reguli pot substitui astfel mecanismele de control ale calculatorului, evident nu la nivelul instrucțiunilor elementare.

În consecință principala funcție a părții de acțiune a regulii este cea de realizare a funcției de comunicare, adică de înscriere a informației pe care regula o produce în context. Se pot aminti principalele operații ce se execută asupra contextului cum sînt:

- ▶ ștergerea unei piese de cunoaștere din context, cu acțiune atît asupra piesei și de cunoaștere cît și asupra informației auxiliare menținută de sistem;
- ▶ înscrierea unei piese de cunoaștere în context cît și declanșarea unor proceduri de evaluare a priorităților pentru plasarea noii piese de cunoaștere în mulțimea ordonată a contextului;
- ▶ modificarea unei piese de cunoaștere este o operație ce pornește de la forma veche a piesei de cunoaștere (informație ce se utilizează la recunoașterea acesteia în context) ca prim operand, al doilea operand fiind dat de o nouă formă a piesei de cunoaștere înscrisă în context pe aceeași poziție, fără a se schimba informația asupra priorității piesei de cunoaștere.

O serie de operații auxiliare sînt posibile, cum ar fi:

- ▶ activarea sau dezactivarea unor reguli de producție, acționînd asupra potențialului de cunoaștere procedurală;
- ▶ alterarea conținutului bazei de reguli, prin ștergerea, înscrierea sau modificarea acestora;
- ▶ inițierea executării unor programe ce au ca efect:
- ▶ execuția unor secvențe de operații ce nu afectează contextul;
- ▶ secvențe de operații ce modifică strategia de control cum sînt apelul explicit al regulii, testări ce folosesc la diverse procese decizionale diferite de cele implementate prin strategia decizională a interpretorului;
- ▶ efectuarea unor calcule prin asignarea de valori unor simboluri.

Datorită caracteristicilor structurale ale acestor operații este uneori necesar să se dispună și de operatori cu caracter constructiv cu ajutorul cărora pot fi construite noi piese de cunoaștere cu structuri adecvate; construcții de liste cu obținerea de contexte locale.

4.2.3.4. Organizarea sistemelor de producții

Metoda de reprezentare a cunoașterii cu ajutorul sistemelor de producții a avut un mare succes datorită simplității regulilor, a modularității acestora, formatului liber al pieselor de cunoaștere, aspectului natural al transpunerii în reguli de acest tip a cunoașterii. Simplitatea principială a metodei se pierde atunci când se elimină deficiențele, determinând o mare complexitate a sistemului.

Comunicația regulilor prin intermediul contextului face ca modificarea conținutului unei reguli să aibă influență doar asupra condiției de aplicabilitate și a concluziei. Sistemele de producții sînt caracterizate de o autonomie ridicată ce face ca regulile să fie tratate ca piese de cunoaștere independente. Un sistem de producții instruit prin reguli și fapte poate să rezolve un anumit tip de probleme într-un interval de tip Dt_0 . Dacă sistemul este instruit să rezolve și alte probleme, aceasta determină ca problema inițială să se rezolve într-un timp $Dt_1 > Dt_0$. Rezultă că în urma acumulării cunoașterii sistemul devine din ce în ce mai neperformant, și deci

$$Dt_n > Dt_{n-1} > \dots > Dt_1 > Dt_0.$$

Modul de lucru cu sistemele de producții este adecvat aplicațiilor în domeniile în care cunoașterea este difuză, secvențe de acțiuni nu pot fi determinate. Sînt puțin avantajate pentru reprezentarea prin această metodă situațiile deterministe exprimabile prin algoritmi sau prin fluxuri predominante de control.

4.2.3.5. Măsuri pentru creșterea performanțelor

Utilizarea sistemelor de producții face necesară o temeinică analiză în scopul prevenirii și eliminării cauzelor ce conduc la deprecierea performanțelor sistemului. Aceste măsuri au în vedere:

- ♦ organizarea contextului pentru a se permite o explorare parțială;
- ♦ organizarea bazei de reguli avînd ca scop eliminarea în procesul inferenței a regulilor neadecvate problemei de rezolvat;
- ♦ înglobarea de funcții auxiliare ce facilitează accesul, selecția, execuția și comunicarea.

Organizarea contextului privește obținerea unor grupuri de piese de cunoaștere adecvate rezolvării unor tipuri predefinite de cunoaștere, adecvate rezolvării unor tipuri predefinite de probleme, context ce va fi denumit *context local*. Învățarea din experiență este o activitate teoretic posibilă dar practic

foarte dificilă. Informația referitoare la tipurile de probleme în care o piesă de cunoaștere este implicată nu este suficientă pentru preselecție, însă folosită pentru preordonarea contextului face ca șansele de reușită să crească simțitor.

Specializarea funcțională a regulilor privește grupări ce se realizează pe baza similitudinii obiectivelor urmărite la aplicarea regulilor de producție individuale, adăugarea de noi contexte prin care sistemul reține actuala informație necesară eficientizării procesului de căutare face să se restrângă căutarea numai la acele entități ce pot candida la încheierea cu succes a fazei. Aceste contexte sînt numite *contexte de referințe*. În faza de testare a condițiilor este util să se cunoască care sînt regulile dintr-o bază de reguli dată, ce se încearcă într-un context dat. Se constituie astfel un context de referințe ce informează asupra producțiilor în ale căror condiții se află o anumită piesă de cunoaștere din contextul global. Fiecare înregistrare în contextul de referințe privind apartenența la partea de condiții va fi legată de o piesă distinctă de cunoaștere din contextul global și va avea structura:

[<piesa de cunoaștere> <lista de reguli>].

Într-un context local specific unei probleme și la o specializare corespunzătoare în baza de reguli pot fi eliminate căutările inutile. Astfel, prin extragerea din contextul de referințe a acelor ce corespund pieselor de cunoaștere din contextul local, reunirea listelor de reguli asociate, obținerea listei regulilor active pentru primul pas al ciclului prin intersecția între lista reunită din pasul precedent și acțiunea specializată pe problema din baza de reguli, eficiența procesului va crește.

Un alt context de referințe poate conține pentru fiecare regulă lista de referință la piesele de cunoaștere existente în partea de condiție cu structura de mai jos

[<regula> <listă piese cunoaștere>].

Utilizarea acestui context presupune intersecția referințelor din context ce corespund regulilor active cu contextul local, după care regulile ce mențin conținutul listei de piese de cunoaștere asociate vor fi selectate pentru faza de corespondență a ciclului. Sistemele ce utilizează contexte de referință obțin performanțe mult mai bune comparativ cu sisteme tradiționale ajungînd de pînă la șase ori mai mari. Componentele interpretorului de reguli ce prelucrează contextele de referințe sînt denumite *filtre*.

Metoda împărțirii bazei de reguli în grupuri independente de reguli de producție specializate, în efectuarea unor acțiuni prestabilite, în soluționarea unor probleme specifice determină noțiunea de *sursă de cunoaștere*. O sursă de cunoaștere acționează independent de celelalte asupra pieselor de cunoaștere și produce date noi numite ipoteze ce sînt fie confirmări, fie obiecte noi. Ipotezele se depun într-o zonă de memorie comună numită și tablă (blackboard). O tablă este împărțită în planuri ce reprezintă zone special dedicate unor categorii de piese de cunoaștere. În cadrul unui plan se pot partiționa nivele ce corespund surselor de cunoaștere distincte.

Construcția bazei de cunoștințe astfel încât procesul inferențial să fie cât mai scurt poate fi realizată după un algoritm ce pornește de la ideea algoritmului ID3 propus de Rose Quinlan. Se dă mai jos exemplul unei baze de cunoștințe specifică unui sistem expert cu aplicații în autodiagnoză, cu învățare din exemple. Se pune problema organizării în forma cea mai adecvată a cunoștințelor. Se generează reguli de clasificare sub formă de arbori de decizie pe baza unui set de exemple. Notînd cu t_1, t_2, t_3 teste care după execuție întorc valori în domenii cunoscute cum sînt (t_{1a}, t_{1b}, t_{1c}) (t_{2a}, t_{2b}, t_{2c}) (t_{3a}, t_{3b}) și d_1, d_2 clase de defecte posibile obținem exemple de defectare ca:

$t_{1a}, t_{2a}, t_{3b} : d_1$
 $t_{1a}, t_{2c}, t_{3a} : d_1$
 $t_{1c}, t_{2a}, t_{3a} : d_2$

Exemplele date mai sus vor fi citite astfel: dacă t_1 întoarce rezultatul t_{1a} , t_2 întoarce rezultatul t_{2a} , t_3 întoarce rezultatul t_{3b} atunci se constată defectul aparținînd clasei defectelor de tip d_1 . Pot fi obținuți astfel mai mulți arbori corecți din punct de vedere logic, dar avînd lungimi diferite în funcție de nodul inițial. Arborii diferă prin ordinea nodurilor de decizie ce reprezintă formal ordinea de execuție a testelor. Pentru micșorarea timpului de atingere a scopului, înălțimea arboreului trebuie să fie cât mai mică. Considerînd următoarele atribute cu valorile lor:

a_1 cu valorile $a_{11}, a_{12}, \dots, a_{1p}$
 a_2 cu valorile $a_{21}, a_{22}, \dots, a_{2k}$

 a_n cu valorile $a_{n1}, a_{n2}, \dots, a_{nj}$

și obiectele, (aici defecte) d_1, d_2, \dots, d_q ce sînt caracterizate de atributele a_1, a_2, \dots, a_n , cantitatea de informație din setul de exemple este dată de relația

$$C = -P_{d1} * \log(P_{d1}) - P_{d2} * \log(P_{d2}) - \dots - P_{dn} * \log(P_{dn})$$

în care, P_{di} reprezintă probabilitatea de apariție a lui d_i în setul valorilor d_1, d_2, \dots, d_n obținîndu-se $P_{di} = \text{numărul de apariții ale lui } d_i / n$.

Atributul a_i împarte prin valorile sale $a_{i1}, a_{i2}, \dots, a_{ik}$ setul de exemple în clase. Dacă există:

$n1$ obiecte cu valoarea atributului $a_i = a_{i1}$
 $n2$ obiecte cu valoarea atributului $a_i = a_{i2}$

și notînd cu n numărul inițial de obiecte, se obține cîștigul de informație dat de alegerea atributului a_i pentru formarea nodului de decizie ca fiind $C - C(a_i)$, în care

$$C(a_i) = (n1/n) * C1 + (n2/n) * C2 + \dots$$

unde C_i reprezintă "cantitatea de informație" în fiecare din clasele de exemple obținute prin partitionarea după un atribut. Atunci se va alege ca atribut inițial de decizie acel atribut pentru care valoarea $C - C(a_i)$ este maximă. Pentru colecția de teste t_1, t_2, t_3 dacă atributul t_2 are cîștig de informație față de t_1 atunci t_2 îl va precede pe t_1 .

Procedura de construcție arbore este ilustrată în pseudocod mai jos.

```
//ex = set exemple inițial
//arb = arbore de decizie obținut
atribuie set (ex, null)
cât timp setul nu este vid repetă
    atribuie m-alege element (set) //m-colecție de exemple cu arborele
    aferent
    dacă contradicție (m) atunci //verifică dacă mulțimea aleasă
    conține //obiecte de tip diferit
    atribuie a--alege-atribut (m) //se alege atributul cu cel mai mare
    "cîștig"
    atribuie ma formează mulțimi (a, m) //se obțin clasele în care se
    împarte //setul de exemple funcție de valoarea lui a
dacă nod = null atunci
    atribuie nod, formează nod (a) //formează rădăcina arborelui
altfel completează nod cu atribut (nod, a)
pentru succesiunea aparținînd ma repeta
    atribuie nod p <- obține nod
    set <- set + (succesorii, nod p)
    completare nod cu fii (nod, nod p)
altfel atribuie nod selectează nod (m)
    completează nod cu mesaj de sfîrșit parcurgere arbore și elementul
    la care s-a ajuns
sfîrșit.
```

Se consideră la acest moment un caz ipotetic de defectare pentru care se exemplifică organizarea după atributul t_1 . Acest atribut împarte setul inițial de exemple în două clase ce sînt caracterizate de rezultatul întors de t_1 , adică t_{1a} și t_{1b} . Clasa caracterizată de t_{1a} este

$t_{1a}, t_{2c}, t_{3b} : d_2$

$t_{1a}, t_{2b}, t_{3a} : d_1$

$t_{1a}, t_{2a}, t_{3a} : d_2$

$t_{1a}, t_{2c}, t_{3a} : d_1$

$t_{1a}, t_{2a}, t_{3b} : d_2$

și clasa caracterizată de t_{1b}

$t_{1b}, t_{2c}, t_{3a} : d_1$

$t_{1b}, t_{2a}, t_{3a} : d_2$

$t_{1b}, t_{2c}, t_{3b} : d_2$

Se va obține pentru cîștigul informațional calculat după relațiile de mai sus:

$$C = -3/8 * \log_2 3/8 - 5/8 * \log_2 5/8 = 0.954$$

$$C1 = -2/5 * \log_2 2/5 - 3/5 * \log_2 3/5 = 0.971$$

$$C2 = -1/3 * \log_2 1/3 - 2/3 * \log_2 2/3 = 0.918$$

$$C(t_1) = 5/8 * 0.971 + 3/8 * 0.918 = 0.951$$

$$C - C(t_1) = 0.003.$$

Dacă clasificarea se face după valorile atributului t_2 se vor obține rezultatele:

$t_{2a}, t_{1a}, t_{3a} : d_2$

$t_{2a}, t_{1a}, t_{3b} : d_2$

$t_{2a}, t_{1b}, t_{3a} : d_2$

clasa caracterizată de t_{2b}

$$t_{2b}, t_{1a}, t_{3a} : d_1$$

și clasa caracterizată de t_{2c}

$$t_{2c}, t_{1a}, t_{3b} : d_2$$

$$t_{2c}, t_{1a}, t_{3a} : d_1$$

$$t_{2c}, t_{1b}, t_{3a} : d_1$$

$$t_{2c}, t_{1b}, t_{3b} : d_2$$

Va rezulta:

$$C1 = -3/3 * \log_2 3/3 = 0.000$$

$$C2 = -1/1 * \log_2 1/1 = 0.000$$

$$C3 = -2/4 * \log_2 2/4 - 2/4 * \log_2 2/4 = 0.500$$

$$C(t_2) = 3/8 * 0.000 + 1/8 * 0.000 + 4/8 * 0.500 = 0.250$$

$$C - C(t_2) = 0.704$$

Este evident că atributul t_2 are un câștig net de informație față de atributul t_1 , fapt pentru care regulile ce se grupează după acest atribut vor trebui să le preceadă pe cele grupate după atributul t_1 .

Alte cunoștințe de o importanță deosebită conținute în baza de cunoștințe se referă la ordinea de execuție a testelor. Întrucât nu se dispune în general de un analizator de semnătură, ce ar fi foarte util pentru testarea magistralelor și a bufferelor, o ordine de execuție a testelor bine gândită va ajuta depistarea defectelor. Spre exemplu dacă este posibil accesul la date pe cele două părți ale magistrei (lo sau hi) se vor testa ambele independent, apoi global coroborînd concluziile. De asemenea, în sistemele cu procesoare multiple, procesorul master va impune ordinea testelor plachetelor subordonate.

4.2.3.6. Inconsistența sistemelor de producții

Cele mai multe aplicații sînt realizate cu sisteme bazate pe reguli de producție din cauză că există o gamă destul de largă de programe cum sînt ESE (IBM), Personal Consultant Plus (Texas Instruments) pentru construcția sistemelor expert bazate pe acest mod de reprezentare. Ele sînt în general succesoare ale sistemului expert EMYCIN și au aceeași structură. O problemă importantă este cea a obținerii sistemelor expert cu o mare cantitate de cunoaștere, situație în care este greu să se mențină consistența bazei de cunoștințe.

Modul în care se trage o concluzie într-un sistem de producții se mai numește și *derivație*. Pentru un sistem de producții $R = (D, P)$ și $R' = (D', P)$

se definește $(D, P) \stackrel{1}{\vdash}_{PS} (D', P)$ dacă și numai dacă există regula IF c THEN t regulă ce aparține lui P cu c component al lui D , de unde rezultă că $D' = D \cup \{t\}$. Se va nota $\stackrel{1}{\vdash}_{PS}$ închiderea tranzitivă și reflexivă.

Definiție: O bază de date D este inconsistentă dacă și numai dacă există termeni $x = a$ și $x \neq a$ ca elemente în D .

Fie ca exemplu o bază de date D a unui sistem de producții cu $D = \{\text{cutie} = \text{mare rezistență}, \text{șurub} = \text{șurub ascuns}, \text{șurub} \neq \text{șurub ascuns}\}$

Cum $\text{șurub} = \text{șurub ascuns}$ și $\text{șurub} \neq \text{șurub ascuns}$ sînt două elemente în D rezultă că D este inconsistența.

Definiție: Un sistem de producții $R = (D, P)$ este inconsistent dacă și numai dacă există o bază de date D' și $(D, P) \stackrel{1}{\vdash}_{PS} (D', P)$, iar D' este inconsistent. Cu alte cuvinte pentru o bază de date consistentă D în sistemul de producții (D, P) se poate obține o bază inconsistentă în urma aplicării derivatei.

Se consideră mai jos inconsistența într-un mod foarte limitat legat de posibilitatea de a deduce contradicții în sistemele de producție. Ca exemplu se pornește de la următorul sistem de producții $R = (D, P)$, în care

$D = \{\text{Construcție} = \text{rezistență}, \text{cutie} = \text{aspectuoasă}, \text{conectare} = \text{puternică}\}$
 $P = \{P_1 : \text{IF } \text{construcție} = \text{rezistență} \text{ cutia} = \text{aspectuoasă} \text{ THEN } \text{confecție} = \text{aluminiiu}$
 $P_2 : \text{IF } \text{cutia} = \text{aspectuoasă} \text{ THEN } \text{șurub} = \text{șurub cap îngropat}$
 $P_3 : \text{IF } \text{conectare} = \text{puternică} \text{ THEN } \text{șurub} \neq \text{șurub cu cap îngropat}\}.$

Desigur că chiar dacă D este consistent se va obține

$D = \{\text{Construcție} = \text{rezistență}, \text{cutie} = \text{aspectuoasă}, \text{conectare} = \text{puternică}, \text{confecție} = \text{aluminiiu}, \text{șurub} = \text{șurub cu cap îngropat}, \text{șurub} \neq \text{șurub cu cap îngropat}\}.$

În sisteme de producții pot fi avute în vedere și alte restricții pornind de la faptul că un parametru poate avea o singură valoare sau mai multe. Restricția prin care un parametru are valoare unică poate fi exprimată ca regulă. Fie $R = (D, P)$ un sistem de producții cu $(R) = \{x_1, \dots, x_m\}$ și $(P) = \{a_1, \dots, a_n\}$ și restricția ca x_k are o singură valoare în P . Se poate asimila sistemul de producții cu sistemul $R' = (D, P')$, unde

$$R' = R \cup \{\text{IF } x_k = a_i \text{ THEN } x_k \neq a_j \mid i, j = \overline{1, n} \text{ și } i \neq j\}$$

Se obține astfel că R este inconsistent dacă și numai dacă $R' = (D, P')$ este inconsistent. În acest mod se poate considera că valorile multiple sînt posibile și în plus se dă un set de reguli pentru parametri ce au o singură valoare.

Consistența sistemelor de producție fără negare poate fi determinată utilizînd algoritmul de mai jos. Fie $R = (D, P)$ un sistem de producții fără negare. Pornind de la baza de date D se generează toți termenii ce pot fi cuprinși în R .

Algoritm:

$D'' = D$

$D' = \text{null}$

while ($D' \neq D''$)

$D' = D''$

$P'' = \{(\text{IF } c \text{ THEN } t) \in P \mid c \text{ este adevărat pentru } D''\}$

$D'' = D'' \cup \{t \mid (\text{IF } c \text{ THEN } t) \in P''\}$

if ($(\exists x = a \ \& \ x \neq a) \in D''$)

then (D, P) este inconsistent

else

(D, P) este consistent.

Finitudea algoritmului se bazează pe faptul că numărul regulilor sistemului de producții este finit.

4.2.4. Rețele semantice

Acest mod de reprezentare a apărut ca o consecință a modului de surprindere a structurilor relaționale de mare complexitate. Aspectul de graf al reprezentării constituie doar o variantă de evoluție pentru caracteristicile puse în relief prin reprezentarea cunoașterii cu predicate de ordinul întâi.

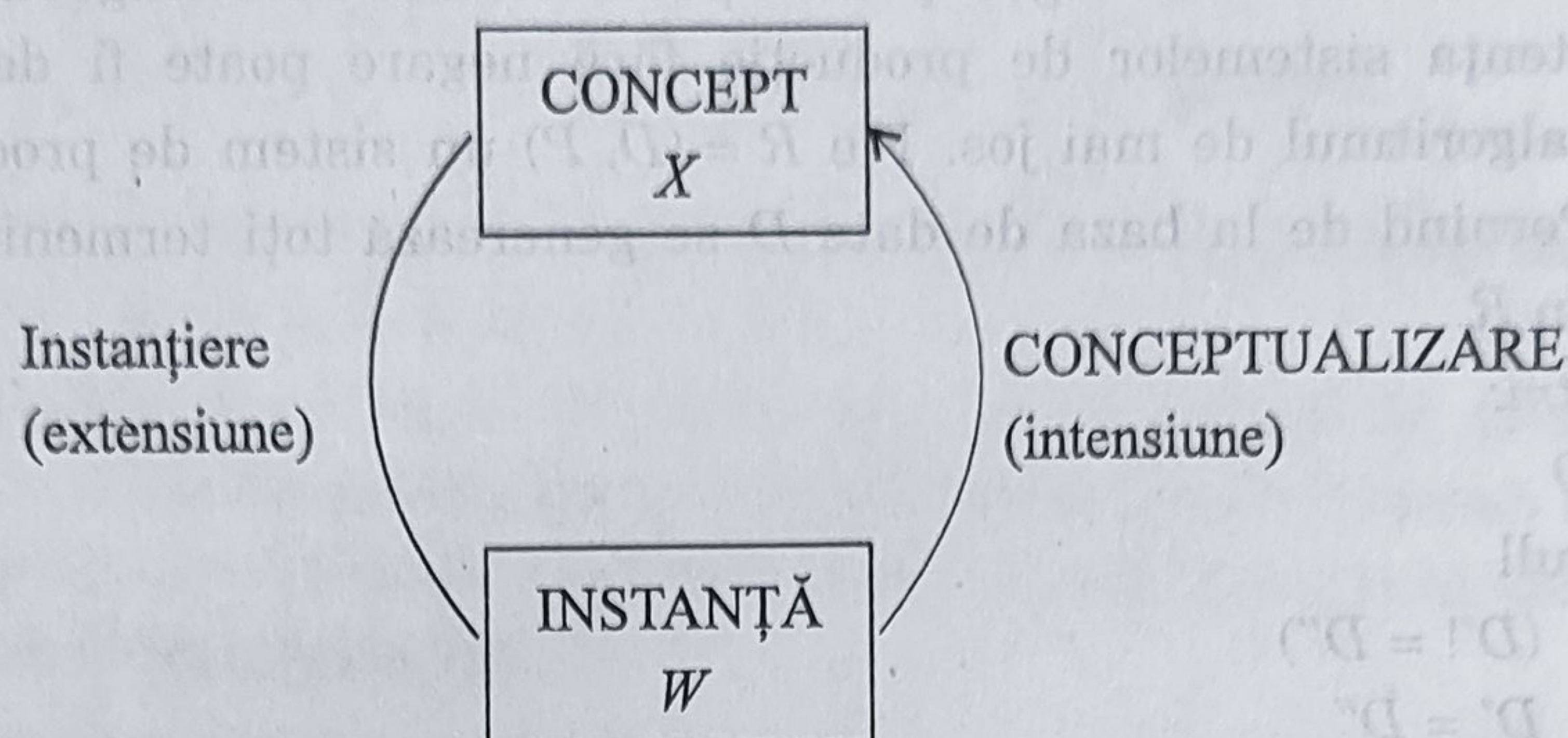
Elementele primitive constitutive ale unei rețele semantice sînt:

► *noduri* - abstractizări structurale pentru concepte, evenimente, stări, alte obiecte suport;

► *arce* (legături) - abstractizări ale relațiilor propriu-zise.

Primitivele de acest fel sînt utilizate în limbajele de nivel înalt ce se folosesc pentru reprezentarea structurilor de date - de spații de memorie numite și celule, legate între ele prin pointeri (ex. LISP). Simbolurile și valorile asociate structurilor de date permit exprimarea unor semnificații și apropierea de cerințele sistemelor de reprezentarea cunoașterii.

▼ Fig. 4.8. Relația concept instantă



Modelul de reprezentare prin rețele semantice a fost propus în 1966 de Ross Quillian și s-a numit "rețea semantică". Denumirea se explică prin faptul că reproduce din punct de vedere conceptual structura semantică a dicționarelor explicative cele două componente fiind:

- ▶ noduri - concepte reprezentate prin cuvinte;
- ▶ arce - relații cu alte concepte.

Denumirea de rețea semantică nu corespunde calităților metodei, motiv pentru care se propune folosirea termenului de rețea asociativă (apropierea de memoria umană). Se menține denumirea de rețea semantică ce cuprinde o varietate de metode ce folosesc structura de rețea pentru construirea bazelor de cunoștințe destinate programelor ce realizează o serie de efecte caracteristice procesului de înțelegere. Se poate utiliza noțiunea de "limbajul unor rețele" ce exprimă concepte, evenimente, stări, episoade și relații între acestea. Pentru limbajul unei rețele se utilizează o serie de simboluri care sînt înțelese de interpretorul rețelei, simboluri ce formează primitivele semantice. Combinarea primitivelor conform regulilor limbajului determină expresii.

Descrierea este efectuată prin *obiecte formale intensionale denumite și concepte* cît și prin *obiecte extensionale denumite instante*. Relația între obiecte și instante este denumită și *instanțiere* sau *denotare* și are caracter extensional pe cînd relația între instante și concepte denumită și *conceptualizare* are un caracter intensional (fig. 4.8).

Aceasta se poate reprezenta cu ajutorul limbajului calculului cu predicate de ordinul întâi:

INTENSIUNE (**CONCEPT** (x), **INSTANTA** (w)) indicînd faptul că un concept este o intensiune a instantei sale.

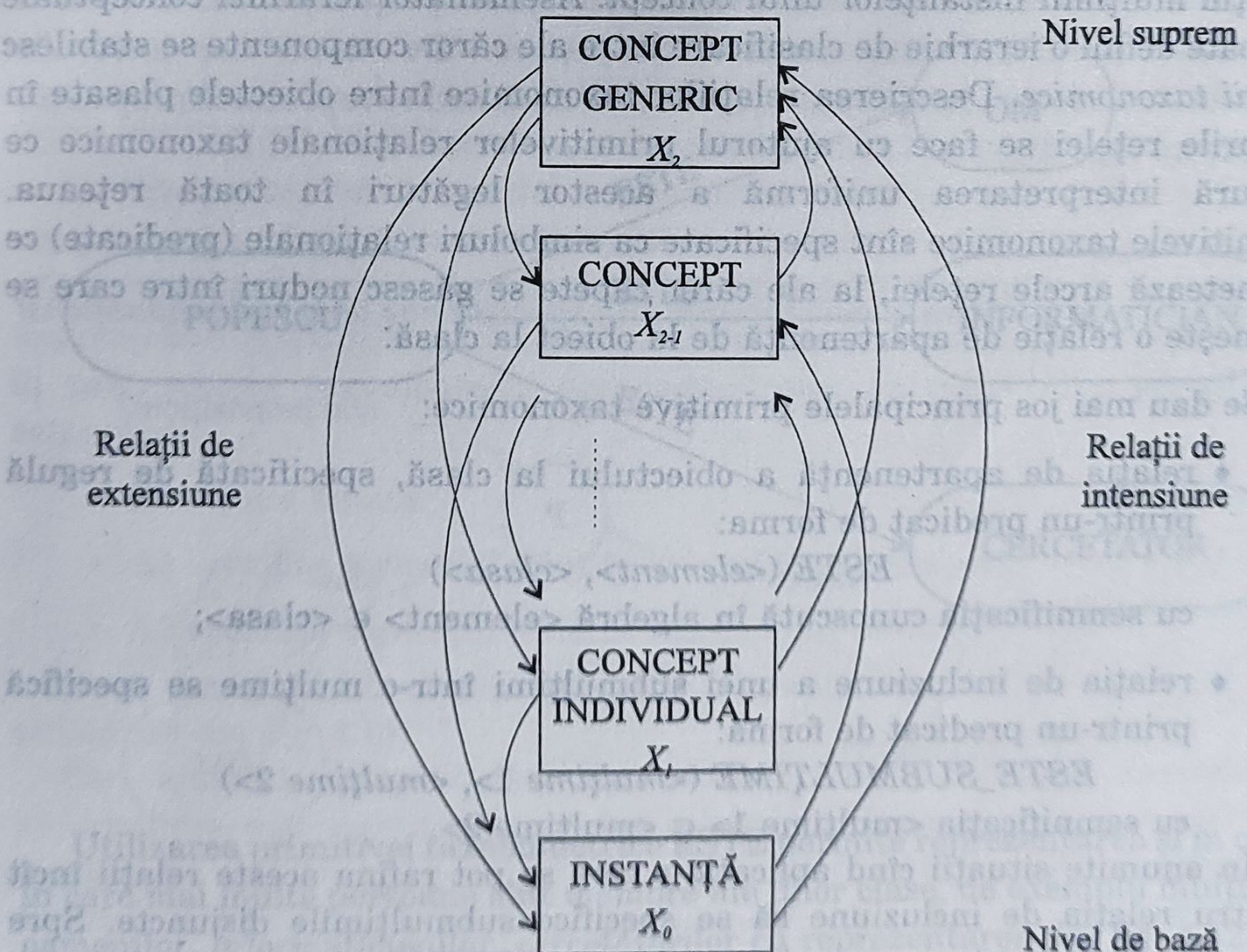
EXTENSIUNE (**INSTANTA** (w), **CONCEPT** (x)) se specifică faptul că o instantă este o extensiune a conceptului, în care:

CONCEPT (x) este funcțional ce ia valori în mulțimea conceptelor.

INSTANTA (w) este un funcțional ce ia valori în mulțimea instanțelor.

O instantă la rîndul său poate reprezenta un obiect din lumea reală sau un alt concept. Atunci cînd instanta este un concept poate avea la rîndul său instanțe. Se obține astfel o ierarhie conceptuală din mai multe nivele. Nivelul de bază al ierarhiei conceptuale este denumit și nivel *infim* și este totdeauna o instantă ce reprezintă un obiect al lumii reale. Toate nivelele unei ierarhii conceptuale sînt concepte generice cu excepția penultimului nivel care este un concept individual, concept ale cărui instanțe sînt indivizi (fig. 4.9).

▼ Fig. 4.9. Relații într-o ierarhie conceptuală



Conceptul fiind de fapt un obiect cu structură este alcătuit din entități formale numite componente împreună cu relațiile dintre acestea. Componentele fiind dependente de concept, natura lor poate fi aceeași cu a conceptului ca părți ale unui întreg, sau diferită de a conceptului ca atribute ale unor obiecte. Orice instanță a unui concept are o structură relațională ca cea specificată pentru conceptul respectiv.

Natura relațiilor poate fi de tipul:

- ♦ *relații epistemice* - permit descrierea semnificației unui concept cu ajutorul unor concepte mai puțin generale;
- ♦ *relații compoziționale* - între concepte și componentele lor;
- ♦ *relații de apartenență* - a conceptelor sau instanțelor la clase sau mulțimi;
- ♦ *relații cauzale* - între concepte de tip obiect și concepte de tip acțiune;
- ♦ *relații modale* - între componente și modalitățile de atașare a acestora la concepte.

De multe ori este greu a se formula un concept dar se pot descrie clase ce aparțin mulțimii instanțelor unui concept. Asemănător ierarhiei conceptuale se poate defini o ierarhie de clasificare între ale căror componente se stabilesc *relații taxonomice*. Descrierea relațiilor taxonomice între obiectele plasate în nodurile rețelei se face cu ajutorul primitivelor relaționale taxonomice ce asigură interpretarea uniformă a acestor legături în toată rețeaua. Primitivele taxonomice sînt specificate ca simboluri relaționale (predicate) ce etichetează arcele rețelei, la ale căror capete se găsesc noduri între care se definește o relație de apartenență de la obiect la clasă.

Se dau mai jos principalele primitive taxonomice:

- ♦ relația de apartenență a obiectului la clasă, specificată de regulă printr-un predicat de forma:

ESTE (<element>, <clasa>)

cu semnificația cunoscută în algebră $\langle \text{element} \rangle \in \langle \text{clasa} \rangle$;

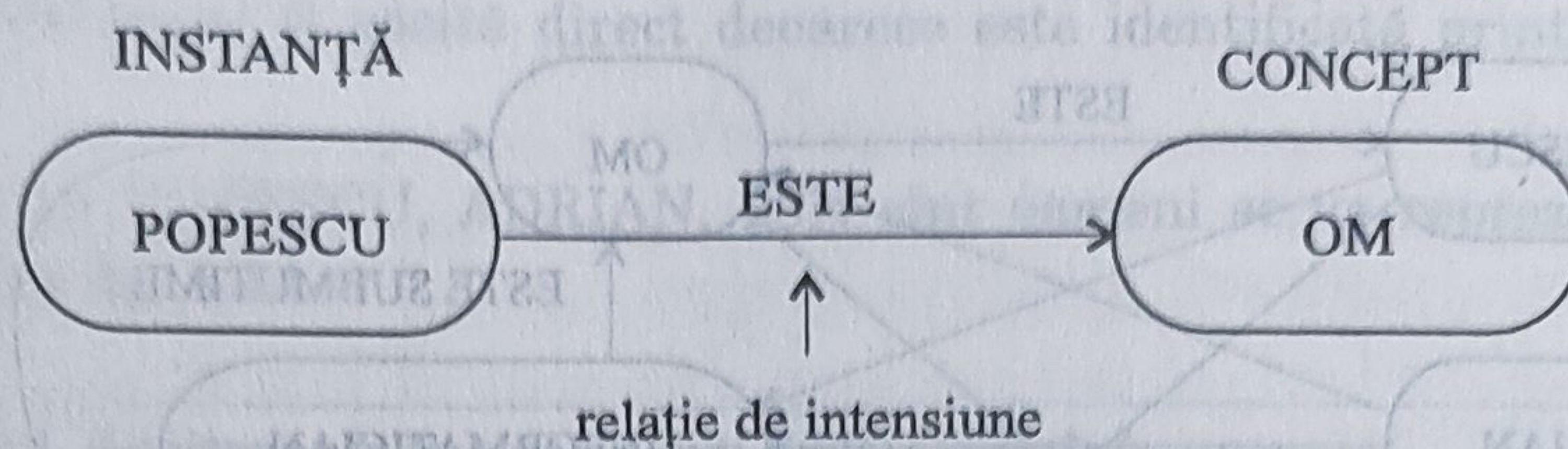
- ♦ relația de incluziune a unei submulțimi într-o mulțime se specifică printr-un predicat de forma:

ESTE_SUBMULTIME (<mulțime 1>, <mulțime 2>)

cu semnificația $\langle \text{mulțime 1} \rangle \subset \langle \text{mulțime 2} \rangle$.

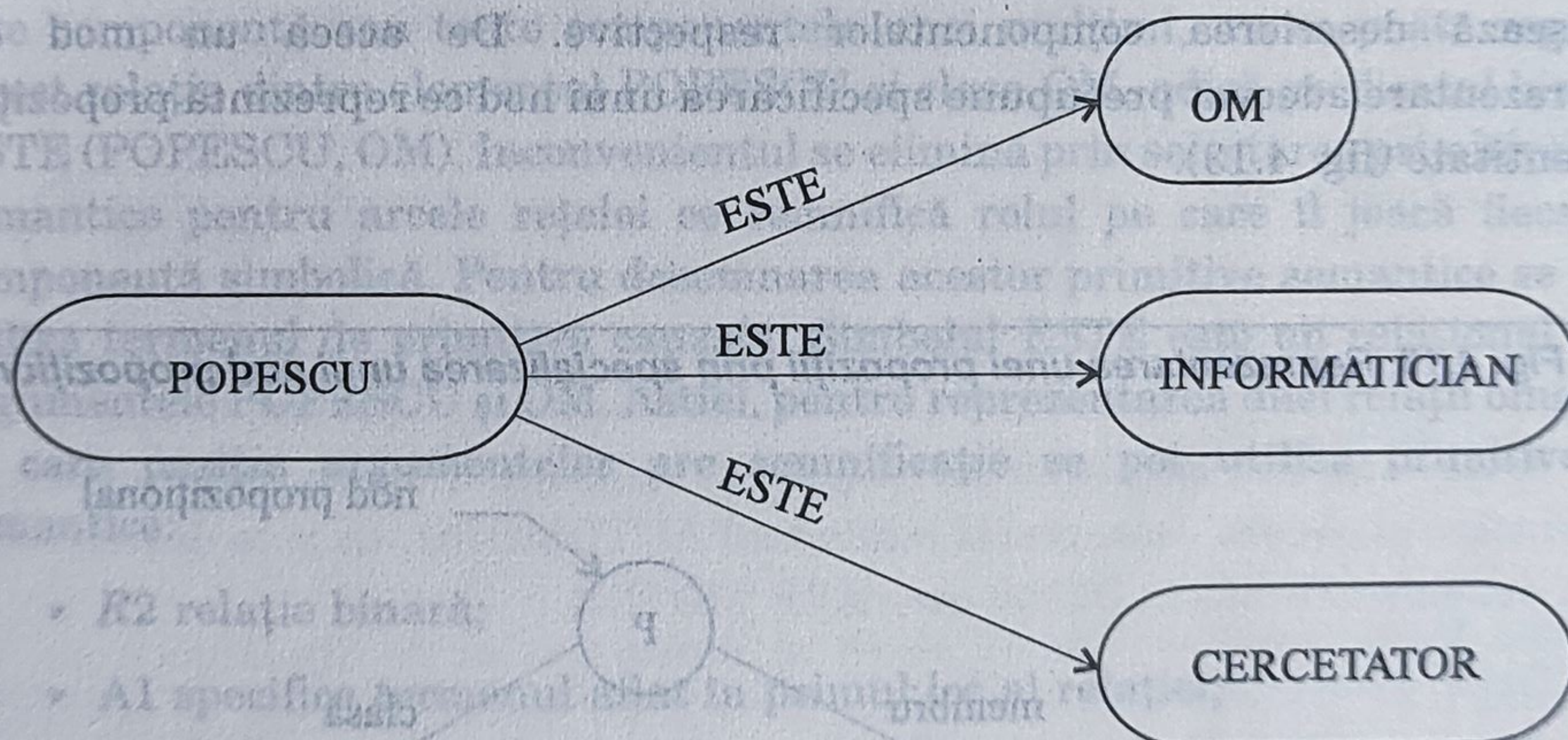
În anumite situații cînd aplicația o cere se pot rafina aceste relații încît pentru relația de incluziune să se specifice submulțimile disjuncte. Spre exemplu apartenența persoanei cu numele POPESCU la clasa oamenilor ce se exprimă cu ajutorul propoziției POPESCU ESTE OM se va transpune într-o rețea semantică cu forma reprezentată în fig. 4.10.

▼ Fig. 4.10. Relația de apartenență



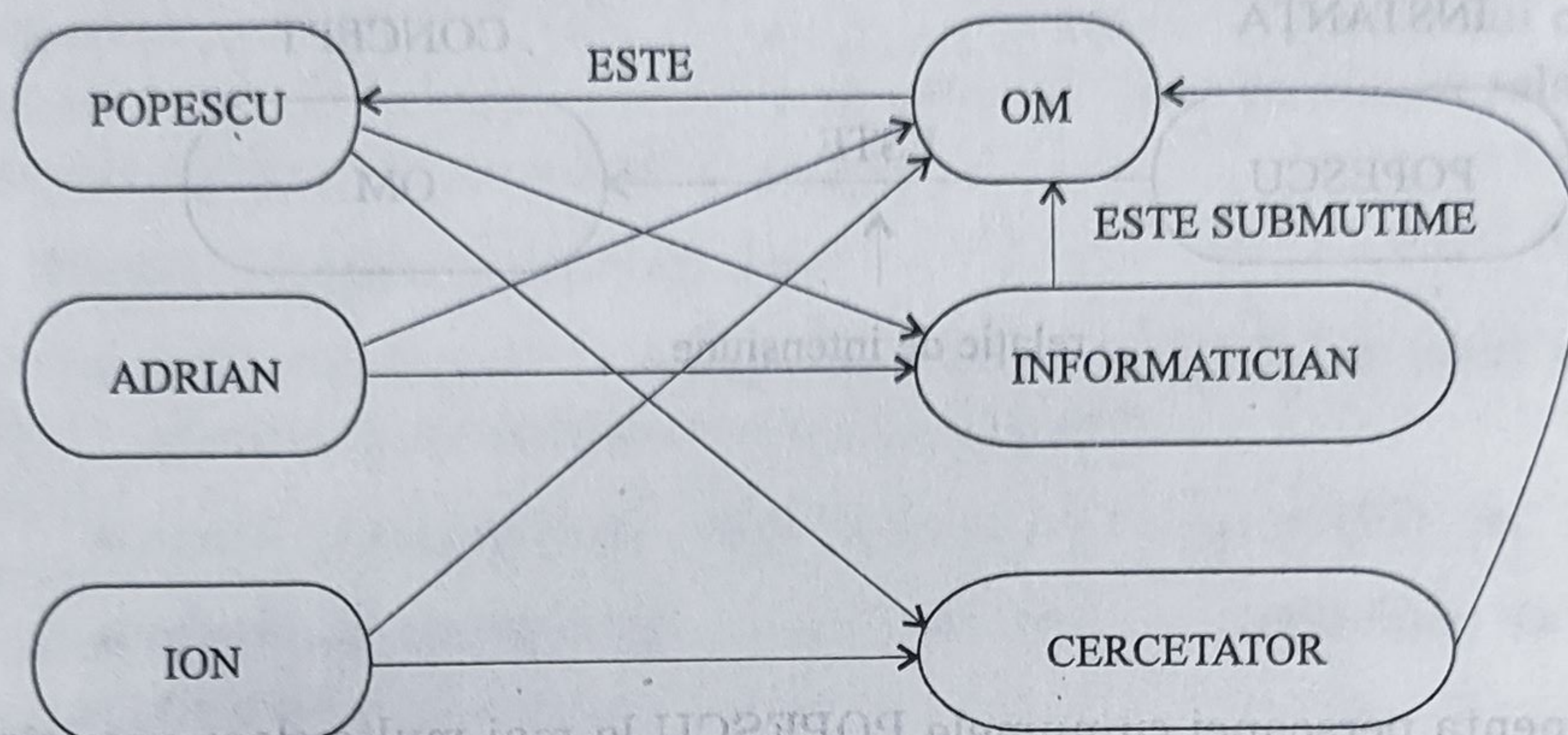
Apartenența persoanei cu numele POPESCU la mai multe clase cum sînt mulțimea informaticienilor, a cercetătorilor și a oamenilor este reprezentată în fig. 4.11.

▼ Fig. 4.11. Utilizarea primitivei taxonomice este.



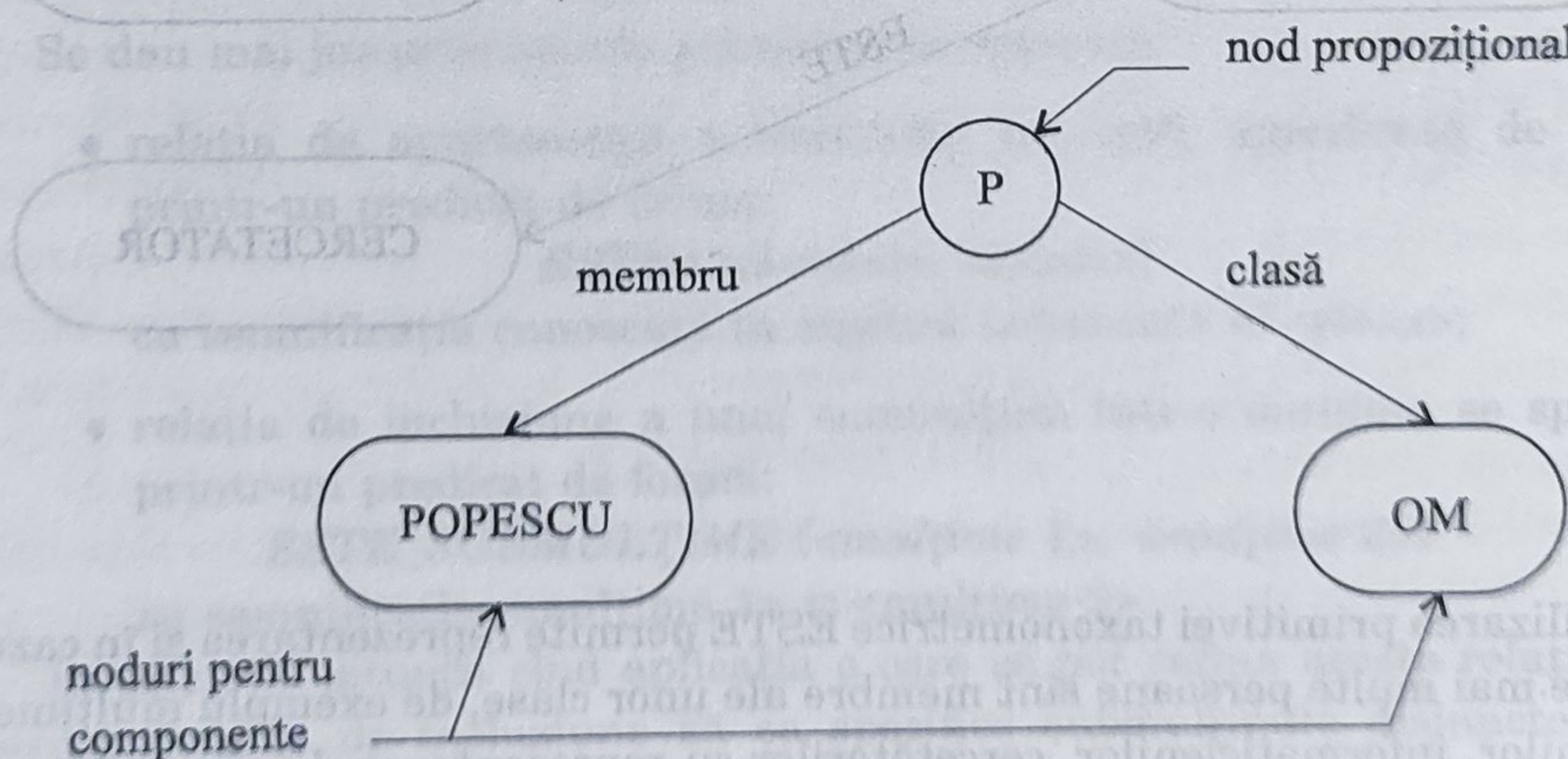
Utilizarea primitivei taxonomice ESTE permite reprezentarea și în cazul în care mai multe persoane sînt membre ale unor clase, de exemplu mulțimea oamenilor, informaticienilor, cercetătorilor cu reprezentarea din fig. 4.12.

▼ Fig. 4.12. Utilizarea primitivei taxonomice este în reprezentarea instanțelor și submulțimilor



Cu toate că reprezentarea este sugestivă este însă dificil de regăsit direct propoziția POPESCU ESTE OM sau altă propoziție corespunzătoare relațiilor specificate, întrucât nu au fost reprezentate noduri propoziționale la care se atașează descrierea componentelor respective. De aceea un mod de reprezentare adecvat presupune specificarea unui nod ce reprezintă propoziția ca entitate (fig. 4.13).

▼ Fig. 4.13. Reprezentarea unei propoziții prin specializarea unui nod propozițional



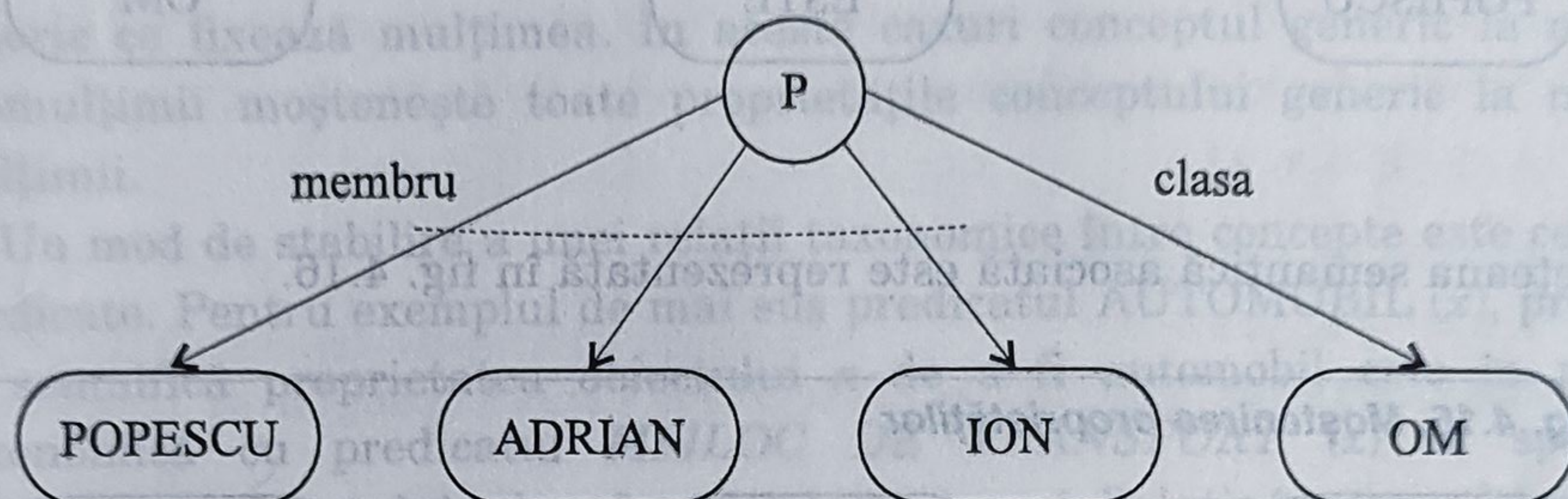
Astfel, propoziția se poate interpreta în forma predicatelor de ordinul întâi prin formula:

$$P = \text{MEMBRU}(\text{POPESCU}) \text{ CLASA}(\text{OM})$$

asertiune ce poate fi găsită direct deoarece este identificată printr-un unic simbol P .

Faptul că POPESCU, ADRIAN, ION sînt oameni se va reprezenta prin rețeaua din fig. 4.14.

▼ Fig. 4.14. Apartenența la clasă a mai multor membri



Chiar dacă în acest mod se pot găsi toate submulțimile la care un element este componentă sau toate componentele unei mulțimi, nu se poate regăsi direct relația dintre elementul POPESCU și clasa OM, adică predicatul binar ESTE (POPESCU, OM). Inconvenientul se elimină prin selectarea primitivelor semantice pentru arcele rețelei ce semnifică rolul pe care îl joacă fiecare componentă simbolică. Pentru desemnarea acestor primitive semantice se va utiliza termenul de primitivă cauzală. Simbolul ESTE este un relațional cu argumentele POPESCU și OM. Astfel, pentru reprezentarea unei relații binare în care poziția argumentelor are semnificație se pot utiliza primitivele semantice:

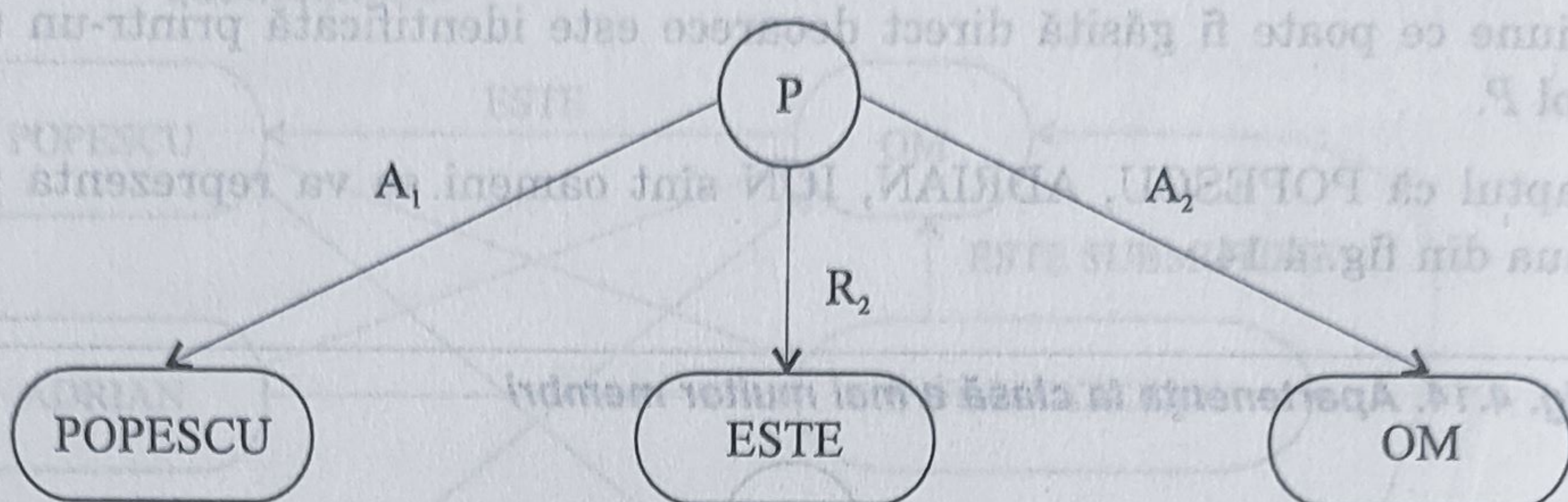
- R2 relație binară;
- A1 specifica termenul aflat în primul loc al relației;
- A2 specifica termenul aflat pe locul doi al relației.

Dacă se reprezintă aceeași aserțiune "Popescu este om" se obține rețeaua semantică din fig. 4.15:

Cu ajutorul primitivelor taxonomice este asigurată moștenirea proprietăților conceptului de către instanțele sale. Spre exemplu, proprietatea conceptului individual AUTOMOBIL de a fi mijloc de transport este moștenită de obiectul D1300 deoarece,

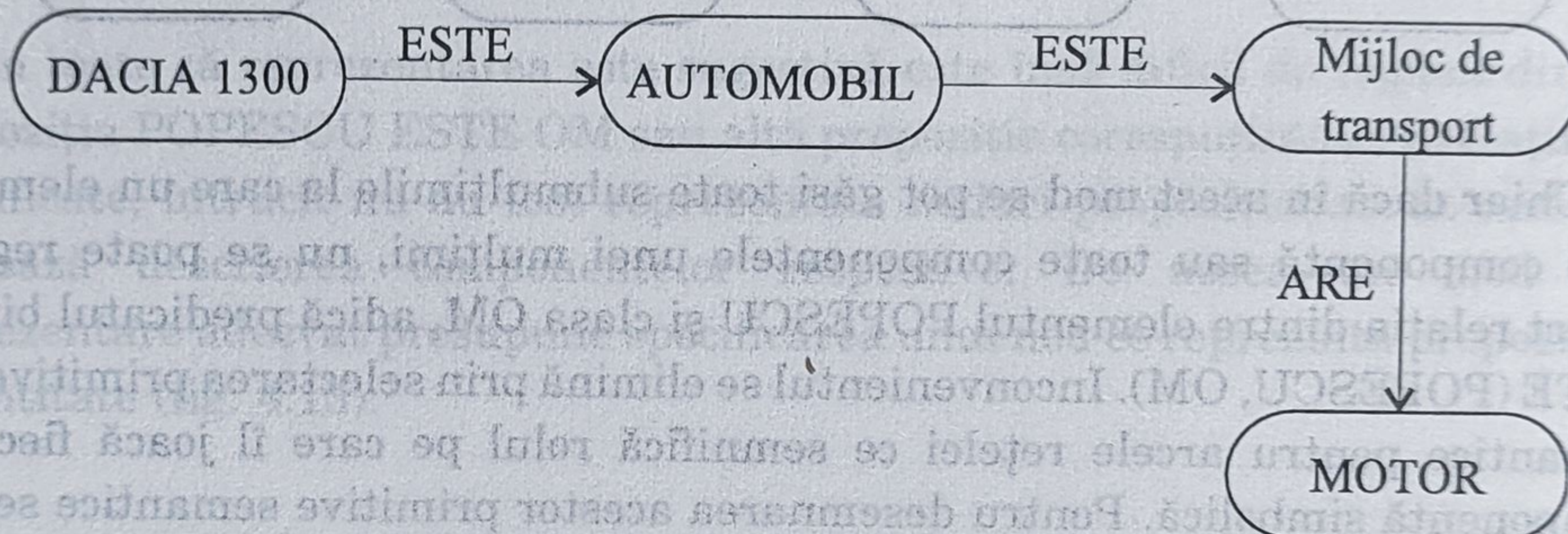
D1300 ESTE AUTOMOBIL
(și) AUTOMOBIL ESTE MIJLOC DE TRANSPORT
Deci, D1300 ESTE MIJLOC DE TRANSPORT.

▼ Fig. 4.15. Reprezentarea unei propoziții utilizând primitive semantice cauzale.



Rețeaua semantică asociată este reprezentată în fig. 4.16.

▼ Fig. 4.16. Moștenirea proprietăților



Proprietatea conceptului de mijloc de transport, de a avea motor se transferă și asupra conceptului individual AUTOMOBIL cât și la descendenții săi extensionali adică, D1300 care ARE MOTOR.

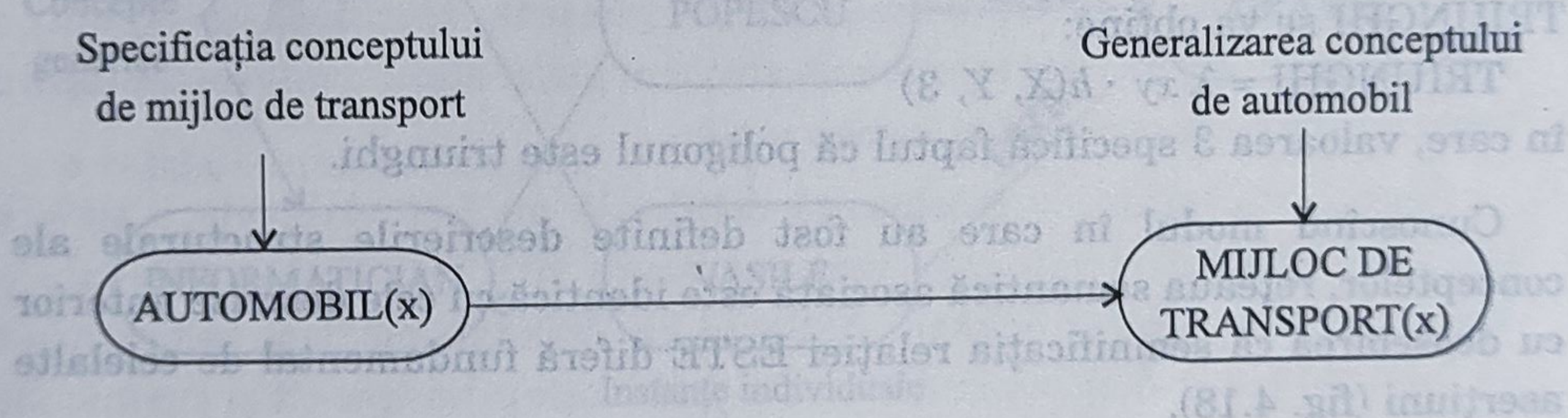
Orice nouă proprietate adăugată ulterior conceptului mijloc de transport se va transmite tuturor descendenților extensionali ai acestuia. De exemplu adăugarea proprietății că mijloacele de transport au sistem de rulare va fi transmisă tuturor automobilelor și în particular pentru D1300. Este necesar a se aminti că relațiile taxonomice se regăsesc în orice rețea semantică, și au inconvenientul folosirii abuzive a singurei primitive semantice ESTE. Relațiile taxonomice pot fi stabilite între următoarele tipuri de noduri: concepte generice, concepte individuale și indivizi. În situația în care nodurile sînt concepte generice de tipul mulțime, relația taxonomică prin primitiva semantică ESTE semnifică incluziunea determinată de un concept generic la mulțimea determinată de alt concept generic indicînd faptul că orice element

al submulțimii este în același timp și element al mulțimii și că toate proprietățile conceptului generic de nivel superior se moștenesc de cele de nivel inferior mai puțin acele proprietăți ce sînt implicate în procesul de constituire a submulțimii.

Pentru conceptul generic AUTOMOBIL se pot instanția submulțimi ale acestuia funcție de tipul motorului cu care este echipat. Dintre aceste submulțimi se pot aminti OTTO, DIESEL, WANKEL, ELECTRIC, TURBOPROPULSOR. În anumite situații criteriul de grupare a elementelor în submulțime nu se regăsește printre proprietățile definitorii ale conceptului generic ce fixează mulțimea. În aceste cazuri conceptul generic la nivelul submulțimii moștenește toate proprietățile conceptului generic la nivelul mulțimii.

Un mod de stabilire a unei relații taxonomice între concepte este cel prin predicate. Pentru exemplul de mai sus predicatul AUTOMOBIL (x), predicat ce semnifică proprietatea obiectului x de a fi automobil este în relație taxonomică cu predicatul MIJLOC DE TRANSPORT (x) ce specifică proprietatea obiectului x de a fi mijloc de transport. Relația între predicate este ilustrată în fig. 4.17.

▼ Fig. 4.17. Relație taxonomică între predicate



Se obișnuiește să se reprezinte în rețele semantice forma predicativă a conceptului fără a se indica variabilele formale atașate. Desigur că o astfel de reprezentare poate genera confuzii referitoare la acceptitudinea în care trebuiesc reprezentate conceptele.

Formula ce reprezintă o generalizare a predicatului AUTOMOBIL (x) prin predicatul MIJLOC DE TRANSPORT (x) este:

$$\forall x \text{ AUTOMOBIL } (x) \rightarrow \text{MIJLOC DE TRANSPORT } (x)$$

Se mai spune că predicatul AUTOMOBIL (x) este o specializare a predicatului MIJLOC DE TRANSPORT (x). O altă interpretare a relațiilor taxonomice între concepte generice în rețelele semantice este λ - abstractizarea conceptelor descrise prin variabilele formale. Variabilele formale în expresiile predicative

ajută la desfășurarea corectă a procesului de substituție. Cu notația uzuală nu se poate distinge cazul în care $f(x)$ se interpretează ca o valoare a funcției pentru o valoare nedeterminată a variabilei x , adică o instantă.

Teoria λ -conversiei introduce notații distincte pentru interpretarea lui $f(x)$ ca funcție de x , cu operatorul de abstractizare funcțională $\lambda x \cdot f(x)$ ce arată că funcția $f(x)$ este de variabilă x cu condiția ca x să apară liber în $f(x)$. Astfel funcția de mai multe variabile $g(x, y, z)$ poate fi interpretată astfel:

- ♦ funcție de o singură variabilă

$$\lambda x \cdot g(x, y, z)$$

- ♦ funcție de două variabile

$$\lambda xy \cdot g(x, y, z)$$

- ♦ funcție de trei variabile

$$\lambda xyz \cdot g(x, y, z).$$

Pentru descrierea generală a conceptului generic poligon se procedează astfel:

$$\text{POLIGON} = \lambda xy n \cdot h(X, Y, n) \text{ în care}$$

$$X = (x_1, x_2, \dots, x_n) \text{ și } Y = (y_1, y_2, \dots, y_n)$$

sînt coordonatele vîrfurilor poligonului și n numărul de vîrfuri.

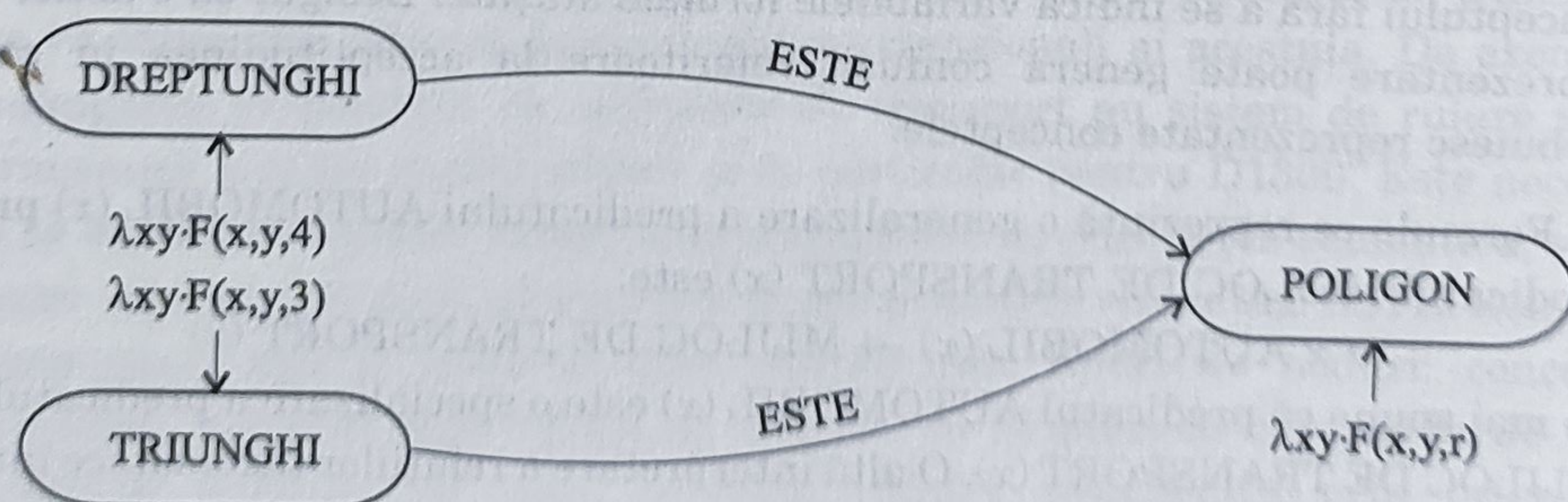
Dacă se dorește dezvoltarea conceptului generic POLIGON spre conceptul TRIUNGHI se va obține:

$$\text{TRIUNGHI} = \lambda xy \cdot h(X, Y, 3)$$

în care, valoarea 3 specifică faptul că poligonul este triunghi.

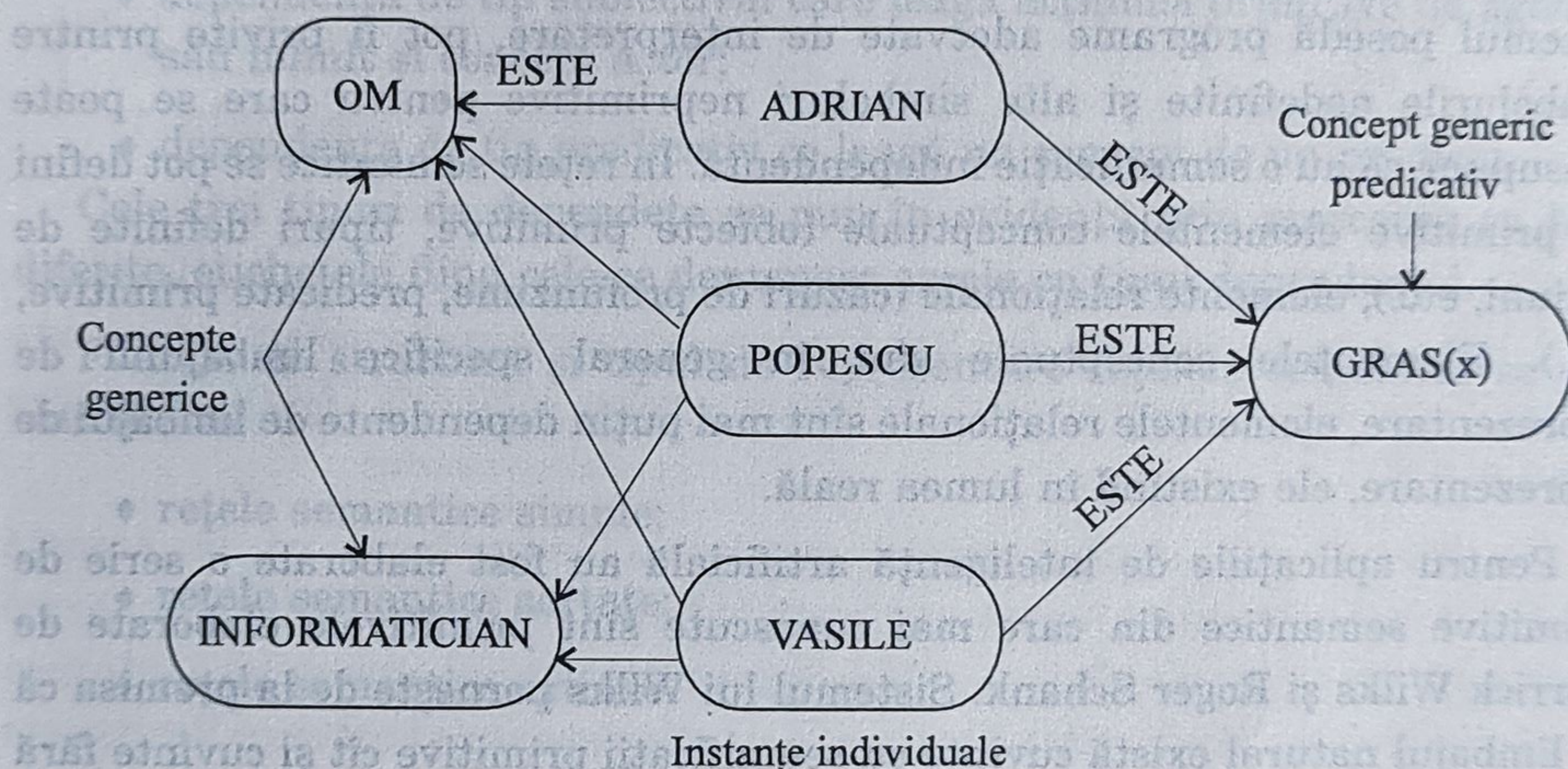
Cunoscînd modul în care au fost definite descrierile structurale ale conceptelor, rețeaua semantică asociată este identică cu cele descrise anterior cu deosebirea că semnificația relației ESTE diferă fundamental de celelalte aserțiuni (fig. 4.18).

▼ Fig. 4.18. Relație taxonomică cu abstractizare.



Rețeaua de mai sus ilustrează modul de instanțiere a conceptelor prin restricționarea valorilor unor variabile ale conceptului generic. Se poate ca abstractizarea să pună în relație concepte generice ce se includ unele în altele. Analizând funcția $f(x, y)$ se observă două concepte generice $x \cdot f(x, y)$ și $y \cdot f(x, y)$ ce sînt în relația λ - abstractizare a conceptului generic $xy \cdot f(x, y)$. Incluziunea conceptelor apare totdeauna cînd un concept generic de nivel inferior are o descriere structurală proprie. Variabilele formale din abstractizarea conceptului de nivel inferior trebuie să se găsească în abstractizarea de nivel superior. Dacă variabilele formale figurează la nivel superior dar nu sînt precizate în abstractizarea de nivel inferior, ele trebuie să figureze în partea descriptivă a funcției fie ca simboluri, fie ca restricții ale acestora.

▼ Fig. 4.19. Relație taxonomică cu predicție.



Interpretarea relației taxonomice între o instanță individuală și un concept pornește în modul cel mai general de la considerarea faptului că instanța este considerată ca element al mulțimii generate de concept. O altă interpretare este dată atunci cînd nodul conceptual specifică un predicat. În acest caz relația ESTE are ca semnificație asignarea instanțelor la variabila formală a predicatului la care i se conferă o valoare de adevăr.

În fig. 4.19 se arată instanțele individuale ADRIAN, POPESCU, VASILE ce au pe lângă proprietățile moștenite de apartenență la clasa de OM, INFORMATICIAN și proprietatea de a fi gras ca urmare a valorilor de adevăr

date de instanțele individuale predicatului GRAS (x). λ - abstractizarea conceptuală poate fi cunoscută pînă la nivelul constantelor sistemului cognitiv, adică la instanțele situate la nivel infim.

A fost reprezentată pînă în prezent cea mai răspîdită primitivă semantică, relația taxonomică, relație ce nu epuizează toate posibilitățile pe care primitiva le oferă reprezentării cunoașterii. Desigur că, nu orice cunoaștere se poate exprima printr-o singură primitivă semantică cu un singur simbol relațional. Instrumentul cel mai performant de reprezentare a cunoașterii este limbajul natural ce dispune de un vocabular vast în continuă evoluție. Dacă prin simboluri de bază înțelegem literele, numărul simbolurilor este limitat, însă combinarea simbolurilor de bază determină diverse simboluri semantice. Problema primitivelor semantice este destul de controversată datorită inexistenței unei definiții unanim acceptate pentru această categorie. Dacă se consideră primitivele semantice ca simboluri nedefinite în limbaj cu semnificația cunoscută de utilizator, pentru care sistemul posedă programe adecvate de interpretare, pot fi privite printre simbolurile nedefinite și alte simboluri neprimitive pentru care se poate presupune că au o semnificație independentă. În rețele semantice se pot defini ca primitive elementele conceptuale (obiecte primitive, tipuri definite de acțiuni, etc.), elemente relaționale (cazuri de profunzime, predicate primitive, etc.). Elementele conceptuale sînt în general specifice limbajului de reprezentare, elementele relaționale sînt mai puțin dependente de limbajul de reprezentare, ele existînd în lumea reală.

Pentru aplicațiile de inteligență artificială au fost elaborate o serie de primitive semantice din care mai cunoscute sînt primitivele elaborate de Yorrick Wilks și Roger Schank. Sistemul lui Wilks pornește de la premisa că în limbajul natural există cuvinte cu semnificații primitive cît și cuvinte fără semnificații primitive. Pornind de la acest fapt Wilks definește următoarele clase de primitive semantice:

- ♦ primitive ce semnifică entități de tip om, lucru, substanță, parte;
- ♦ primitive ce specifică semnificații cu care se reprezintă acțiuni cum sînt cauză, mutare, curgere, lovire;
- ♦ primitive ce specifică semnificații cu care se reprezintă relații între entități de tip obiect, instrument;
- ♦ primitive ce specifică semnificații cu care se reprezintă calificative ale entităților și acțiunilor cum sînt mult, puțin, bun, rău, mic, mare;
- ♦ primitive ce specifică semnificații cu care se reprezintă tipuri de entități;

- ♦ primitive ce specifică clase de primitive, obiecte fizice, oameni, obiecte în interior, exterior.

Pentru cuvintele ce nu sînt primitive se dau definiții cu ajutorul primitivelor semantice. Primitivile semantice trebuie să fie independente în sensul că nu se pot defini cu ajutorul altor primitive semantice asigurînd astfel consistența, în sensul că nu sînt definite circular.

Sistemul lui Schank sau teoria dependențelor conceptuale folosește reprezentarea fără ambiguități a semnificației utilizînd 11 acțiuni elementare grupate în categorii. Într-o astfel de rețea semantică conceptele sînt legate prin relații numite dependente, ce se disting prin tip și etichetă. Pot exista mai multe tipuri de dependențe cum sînt:

- ♦ dependență de tip causal materializată prin descrierea relațiilor conceptuale ale obiectului și legătura sa cu actul primitiv;
- ♦ dependență de tip subiectival care leagă acțiunea primitivă de agentul său numit și concept actor;
- ♦ dependența de tip predicativ ce leagă un concept de un predicat.

Cele trei tipuri de dependențe se pun în evidență prin marcarea cu linii diferite, etichetele fiind cele ce denumesc arcele cu tipul dependenței.

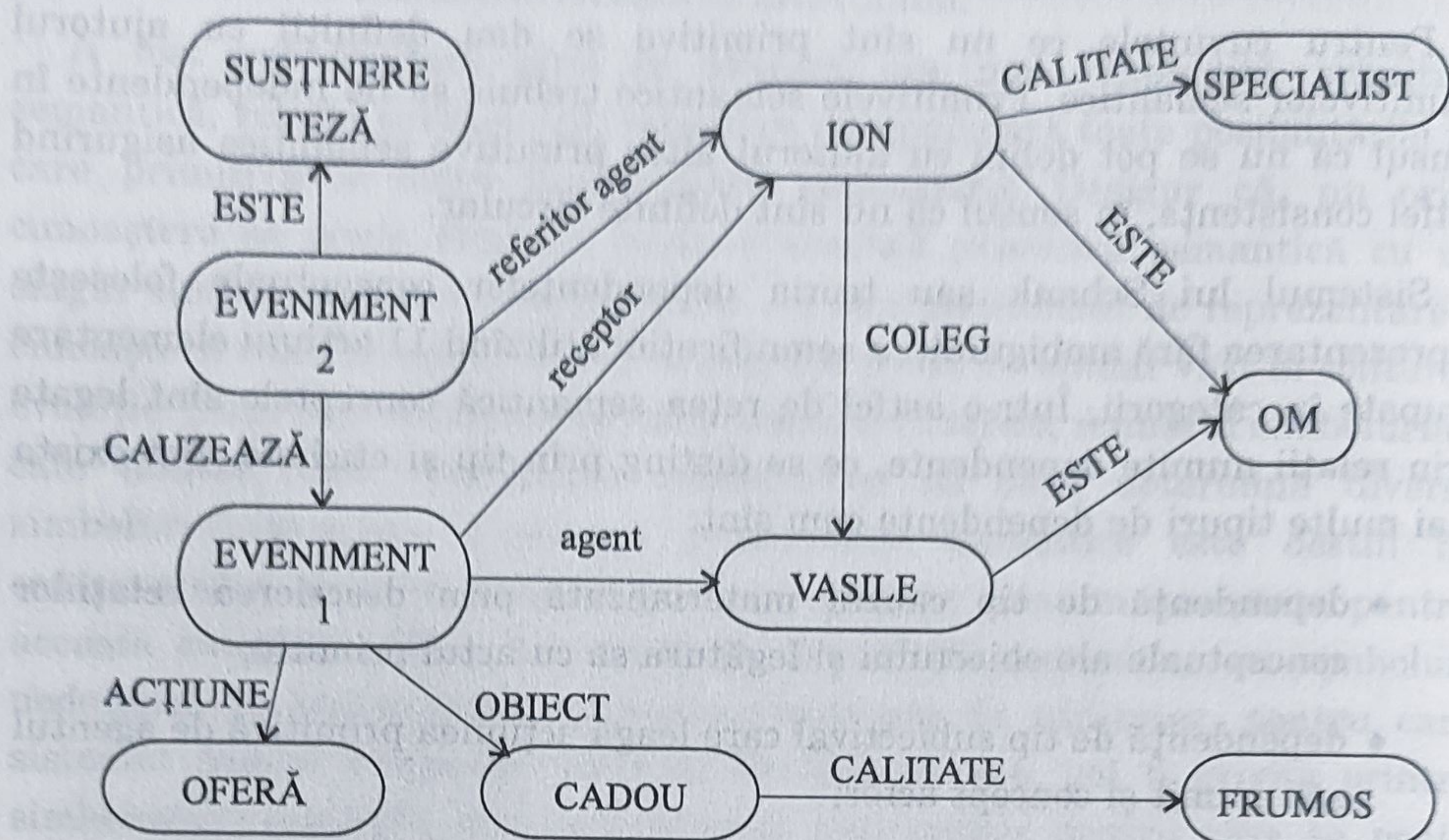
Funcție de structură și tipul de reprezentare rețelele semantice se pot clasifica în:

- ♦ rețele semantice simple;
- ♦ rețele semantice sortate;
- ♦ rețele semantice extinse.

4.2.4.1. Rețele semantice simple

Rețelele semantice simple sînt acele rețele în care nodurile sînt entități individuale iar arcele relații ale acestora. În aceste rețele nu se introduc specializări ale arcelor și nodurilor, fapt ce determină că interpretarea este dată de conținutul descrierii ce este asociat fiecărui simbol ce figurează ca etichetă de nod sau arc. La interpretare se asociază fiecărui simbol de etichetă o procedură proprie de determinare a semnificației, singura trăsătură comună a elementelor rețelei fiind legată de structură și se manifestă sub formă de proceduri fizice de parcurgere a rețelei. Cu ajutorul rețelelor semantice simple se pot reprezenta aserțiuni libere de orice variabilă întrucît nu se admit noduri specializate aferente reprezentării variabilelor.

▼ Fig. 4.20. Rețea semantică simplă



Se presupune reprezentarea piesei de cunoaștere ce specifică faptul că *"la susținerea tezei de doctorat de către Ion care este un bun specialist, colegul său Vasile îi oferă un frumos cadou"*. Se obține o rețea semantică ce are structura din fig. 4.20.

Se observă din studiul rețelei semantice de mai sus că în această reprezentare aserțiunea nu este realizată în forma propozițională. Se cunoaște faptul că Ion și Vasile sînt oameni adică aparțin clasei oamenilor. Acțiunea de oferire a cadoului de către Vasile este un eveniment pentru care Vasile este agent și Ion este receptor. Pentru Ion susținerea tezei de doctorat este de asemenea un eveniment. Acest eveniment este în relație cauzală cu evenimentul oferirii cadoului. Rețelele semantice simple nu oferă noduri preferențiale în inițierea procesului de interpretare, selecția unui nod ca nod inițial este determinată de aplicația pentru care este construită baza de cunoștințe. Candidații pentru poziția de nod inițial sînt selectați după diverse criterii dintre care cele mai întîlnite sînt:

- ♦ nodul etichetat cu un simbol identic numelui piesei de cunoaștere;
- ♦ selecția candidaților dintre nodurile ce au proprietatea de a fi noduri sursă pentru toate relațiile în care sînt implicate.

Pentru descrierea unei rețele semantice simple aceasta este descompusă în triplete de forma

(<nod_sursă> <relație> <nod_destinație>).

Descompunerea sugerează utilizarea de liste în LISP. În rețeaua semantică din fig. 4.20 singurul nod ce este numai nod sursă este nodul <Eveniment_2> ce poate fi considerat nod inițial, lista pornind descrierea de la acest nod este:

(Eveniment_2 este susținerea tezei)

(Eveniment_2 referitor Ion)

(Ion este om)

(Ion calitate specialist)

(Ion coleg Vasile)

(Vasile este om)

(Eveniment_2 cauzează eveniment_1)

(Eveniment_1 acțiune oferă)

(Eveniment_1 obiect cadou)

(Cadou calitate frumos)

(Eveniment_1 receptor Ion)

(Eveniment_1 agent Vasile).

O altă modalitate de reprezentare a rețelei constă în construirea conceptului pentru fiecare nod sursă și asocierea acestuia cu perechi (<relație> <nod destinație>).

((Eveniment_2 (este susținerea tezei)

(referitor Ion)

(cauzează eveniment_1))

(Ion (este om)

(calitate specialist)

(coleg Vasile))

(Vasile (este om))

((Eveniment_1 (acțiune oferă)

(obiect cadou)

(receptor Ion)

(agent Vasile)

((Cadou (calitate frumos)).

Metoda are avantajul eliminării redundanțelor ce apar la specificarea nodului sursă în toate tripletele ce descriu relațiile sale cu nodurile destinație.

4.2.4.2. Rețele semantice sortate

Acest tip de rețele sînt caracterizate prin faptul că folosesc o specializare a nodurilor și arcelor indicînd prin aceasta diferențe de natură structurală ce nu pot fi rezolvate prin etichetare. Cel mai simplu tip de etichetare este cel prin care se specifică un nod de tip propoziție. În elaborarea tehnicilor de etichetare a nodurilor și tipurilor de relații se cunosc mai multe abordări dintre care mult utilizată este cea definită de Fillmore. Fillmore stabilește un număr restrîns de relații primitive ce pot acoperi o varietate semantică largă. O parte dintre acestea sînt:

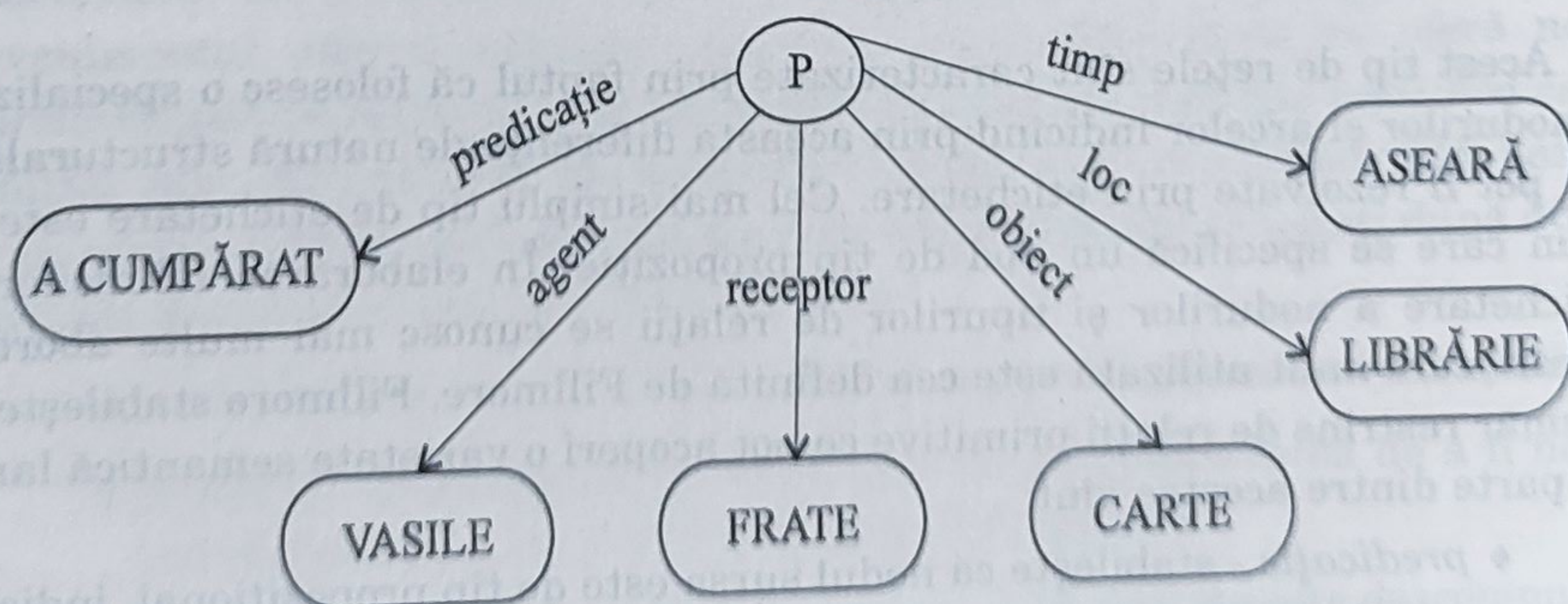
- ♦ *predicație* - stabilește că nodul sursa este de tip propozițional, indicînd o instanțiere a predicatului specificat în eticheta nodului destinație ce reprezintă o instanțiere sau o stare (echivalentă cu verbul într-o propoziție);

- ♦ *agent* - are ca nod destinație o entitate cu capacitatea de a efectua acțiunea. Entitatea este numită și "actor" și are rolul de subiect logic;
- ♦ *receptor* - indică un nod destinație ce are capacitatea de a participa la acțiune;
- ♦ *obiect* - indică ca nod destinație o entitate ce este afectată de acțiunea specificată pentru propoziția din nodul sursa;
- ♦ *instrument* - are ca nod destinație o entitate ce este folosită de agent la efectuarea acțiunii suportate de obiect;
- ♦ *loc* - cu nod de destinație o entitate exprimând o propoziție ce indică locul desfășurării acțiunii;
- ♦ *timp* - indică un nod destinație ce specifică timpul sub o anumită formă. Poate fi exprimată durata prin două relații momentane (*moment_inițial*, *moment_final*), fie ca o relație durată ce arată valoarea intervalului fără a se specifica scara timpului.

Se consideră ca exemplu piesa de cunoaștere *aseară Vasile a cumpărat de la librărie o carte pentru fratele său*, ce se reprezintă sub forma propozițională într-o rețea semantică sortată ca în fig. 4.21, în care se folosesc relațiile cauzale menționate.

Predicația este reprezentată cu ajutorul unui simbol predicativ, cu o aritate corespunzătoare descripției, cu argumentele plasate în două clase și anume *argumente de bază* obligatorii pentru a defini o instanță valabilă a predicatului și *argumente periferice* sau opționale în instanțieri ce aduc în rețea relații suplimentare prin care se conturează cunoașterea sub aspecte ce nu depind exclusiv de predicația considerată.

▼ Fig. 4.21. Reprezentarea propozițională.



Fie $P((x_1, x_2, \dots, x_p), (y_1, y_2, \dots, y_q))$ expresia predicativă generatoare a propozițiilor ce sînt formate cu un verb. O predicatie validă are aritatea n cu $p \leq n \leq p + q$. Expresia $P(x_1, x_2, \dots, x_p)$ se mai numește și predicatie nucleară iar expresia $P((x_1, x_2, \dots, x_p), (y_1, y_2, \dots, y_q))$ se mai numește și predicatie extinsă. Se dau mai jos cîteva sorturi de expresii predicative:

- ♦ sortul expresiilor predicative verbale (V) ce apar în instante ca predicate ale propozițiilor;
- ♦ sortul expresiilor predicative adjectivale (A) ce apar în instante ca modificatori de termeni;
- ♦ sortul expresiilor predicative nominale (N) ce apar în instante ca definiții de termeni.

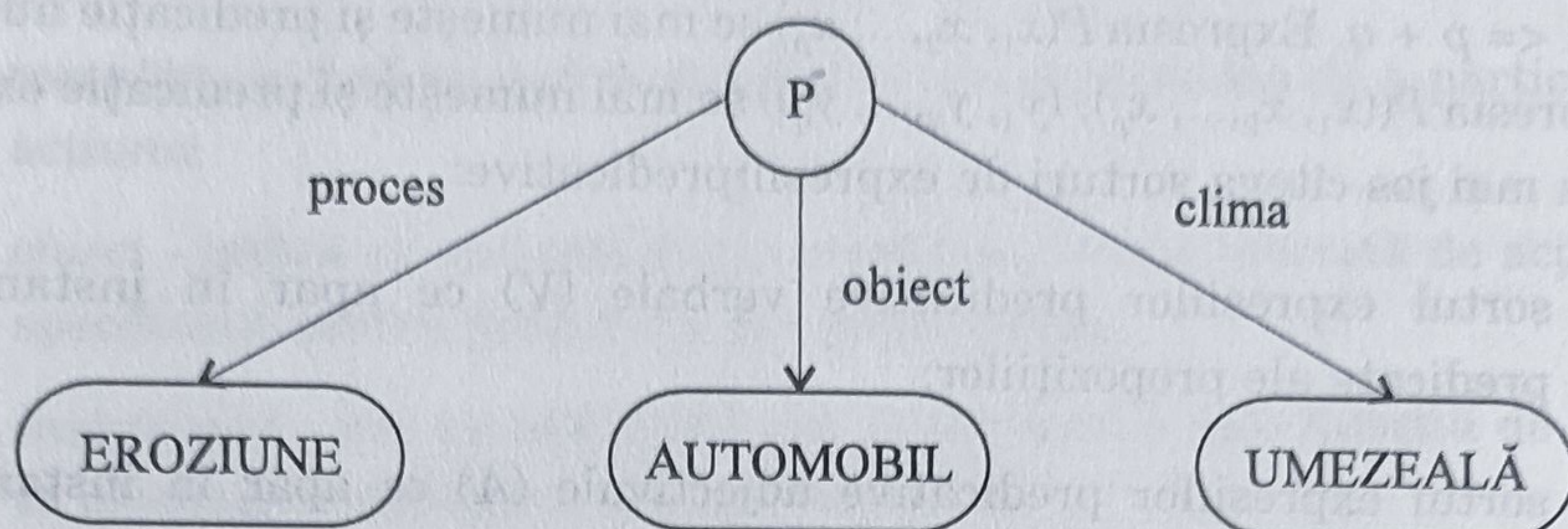
Cadrul definitoriu pentru predicate cuprinde următoarele date inițiale:

- formatul predicatului compus din numele identificator, lista argumentelor de bază și a argumentelor periferice;
- sortul predicatului;
- funcția semantică a argumentelor exprimate sub forma relațiilor cauzale;
- restricții de selecție a termenilor ce candidează pe poziția de argumente ale predicatului.

Starea de lucruri descrisă de o predicatie dă o valoare de adevăr predicatului. Stările de lucruri ce nu se modifică în timp se numesc statice spre deosebire de cele dinamice ce sînt caracterizate de schimbări în timp. Dacă în cadrul predicatului există un argument numit controlor ce are proprietatea de a determina starea de lucruri asociată predicatiei se spune că starea este controlată. Dinamismul și controlul determină noi sorturi ale predicatiei și anume:

- ♦ *acțiunea* - stare dinamică și controlată;
- ♦ *procesul* - corespunde unei stări de lucruri dinamice și necontrolate;
- ♦ *poziția* - corespunde unei stări de lucruri statice și controlate;
- ♦ *situația* - corespunde unei stări de lucruri statice dar necontrolate.

▼ Fig. 4.22. Reprezentarea unei predicatii din sortul acțiune



Pentru predicatele din sortul acțiune, relațiile cauzale de bază determină argumentele părții nucleare cum sînt agentul și obiectul. Relațiile cauzale periferice specifică receptorul acțiunii, direcția desfășurării acțiunii, instrumentul folosit, sursa obiectelor acțiunii, locul și timpul desfășurării. Predicatele din sortul proces nu au relație cauzală de tip agent întrucît sortul este necontrolat, singura relație cauzală de bază este obiectul afectat. Relațiile periferice sînt cele ce specifică relații cauzale ce apar la sortul acțiune. Spre exemplu procesul eroziunii automobilului de către umezeală (fig. 4.22), cu toate că există o cauză a procesului de eroziune, asupra umezelii nu se exercită nici un control, motiv pentru care a fost menționată în relația cauzală clima ca fiind cea care produce acest proces.

Predicatele din sortul poziție posedă un agent ce controlează poziția, și se va nota cu termenul poziționator. Cu o relație de acest tip se pot reprezenta stări de lucruri în care poziționatorul se poziționează pe sine.

Relațiile cauzale se pot extinde fără restricții putînd fi definite următoarele tipuri de relații:

- ♦ relații ce extind specificarea stării de lucruri: calitate, modalitate;
- ♦ relații ce referă predicatia la alți participanți: inițiator, cunoscător, executor, terminator;
- ♦ relații ce extind dimensiunile spațiale: sub, lîngă, deasupra;
- ♦ relații ce extind dimensiunile temporare: durata, frecvența;
- ♦ relații de interpoziționare între o stare și alte stări: cauză, motiv, scop, rezultat, circumstanță.

Sorturile analizate fac posibilă reprezentarea cunoașterii factuale ce implică reprezentarea obiectelor a căror complexitate este limitată la reprezentarea propozițiilor simple. Pentru cunoașterea conceptuală ce implică obiecte abstracte cum sînt variabilele, se introduce un nou sort de nod

corespunzător acestor simboluri. Cu aceste noduri se reprezintă obiecte conceptuale participante la aserțiuni ce îndeplinesc și rolul de conectare a propozițiilor. Un exemplu de acest tip apare în reprezentarea piesei de cunoaștere

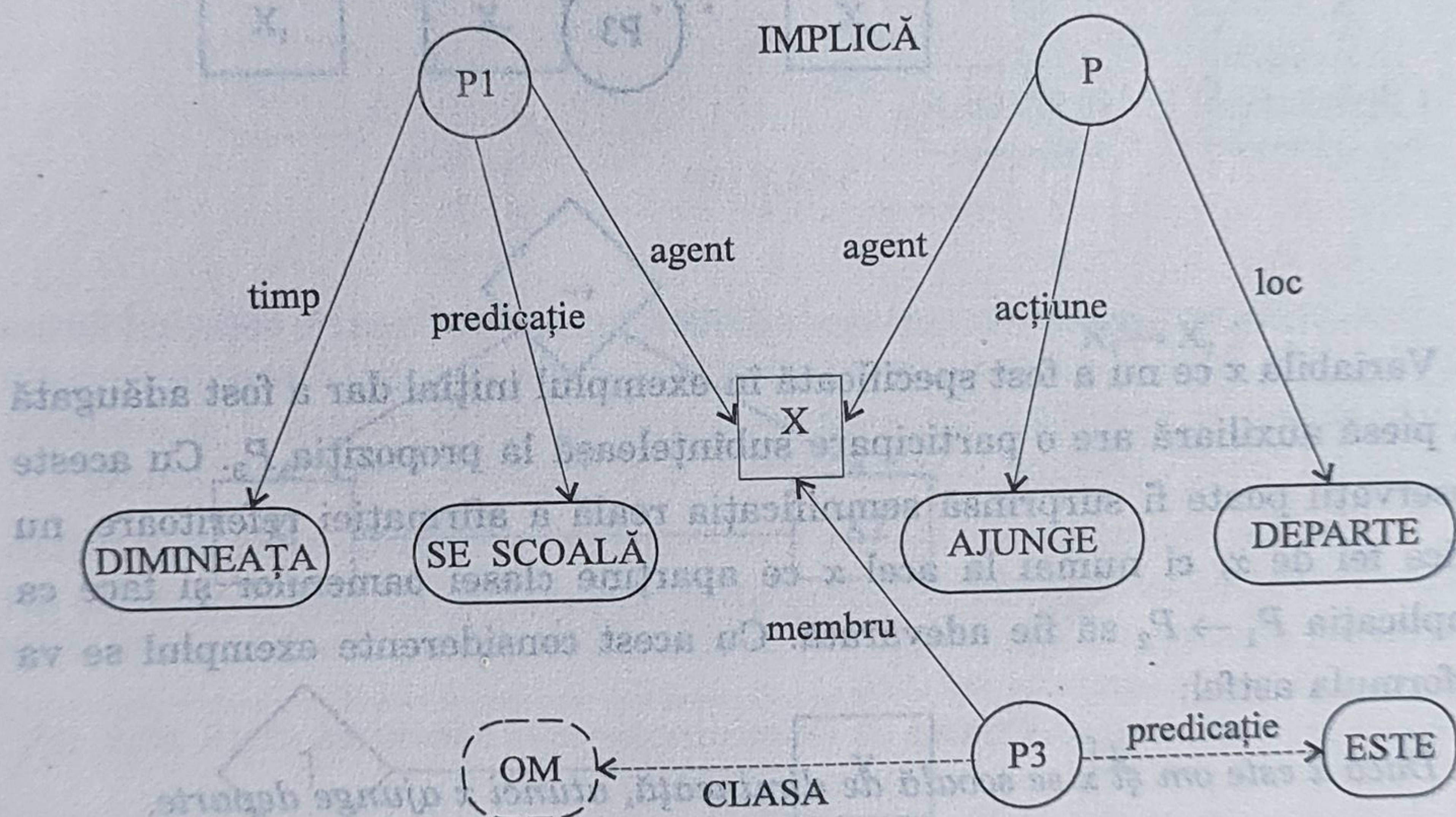
cine se scoală de dimineată, departe ajunge.

În această propoziție cine este echivalent cu orice om, expresie ce se poate scrie cu formula:

$$\forall x (\text{este}(x, \text{om}) \wedge \text{se_scoală}(x, \text{dimizeată}) \rightarrow \text{ajunge}(x, \text{departe})).$$

Reprezentarea acestei formule în rețeaua semantică, presupune reprezentarea relației taxonomice *x este om* ce introduce relațiile cauzale de membru și clasa, și au aceleași funcții semantice ca și relațiile cunoscute *agent* și *obiect*, atunci când se specifică o apartenență la clasă. Pentru această formulă se obține rețeaua semantică din fig. 4.23.

▼ Fig. 4.23. Conectarea rețelelor semantice prin intermediul nodurilor de variabile

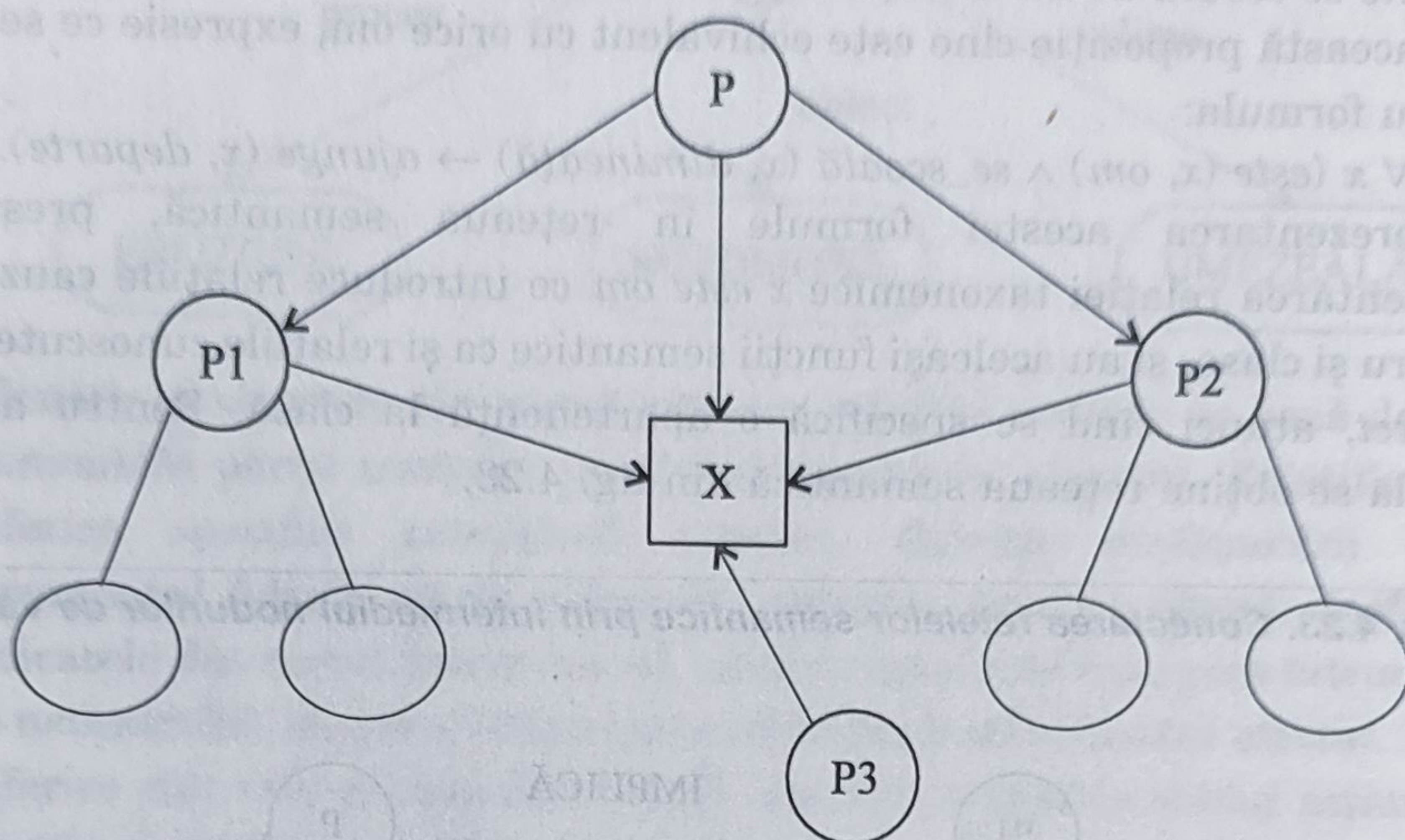


În această rețea s-a dat o rezolvare implicită introducerii cuantificatorilor, prin faptul că expresia reprezentată utilizează un singur cuantificator universal. Dacă rețeaua are o dimensiune mai mare, se introduc cuantificatorii explicit. Implicația poate fi interpretată nu numai ca o relație interpropozițională ci și ca o propoziție de tipul

$$P = \text{oricare}(x) \text{ dacă } (P_1(x)) \text{ atunci } (P_2(x))$$

ce conduce la reprezentarea din fig. 4.24, în care se reprezintă doar propozițiile și cuantificatorul.

▼ Fig. 4.24. Utilizarea explicită a cuantificatorului universal

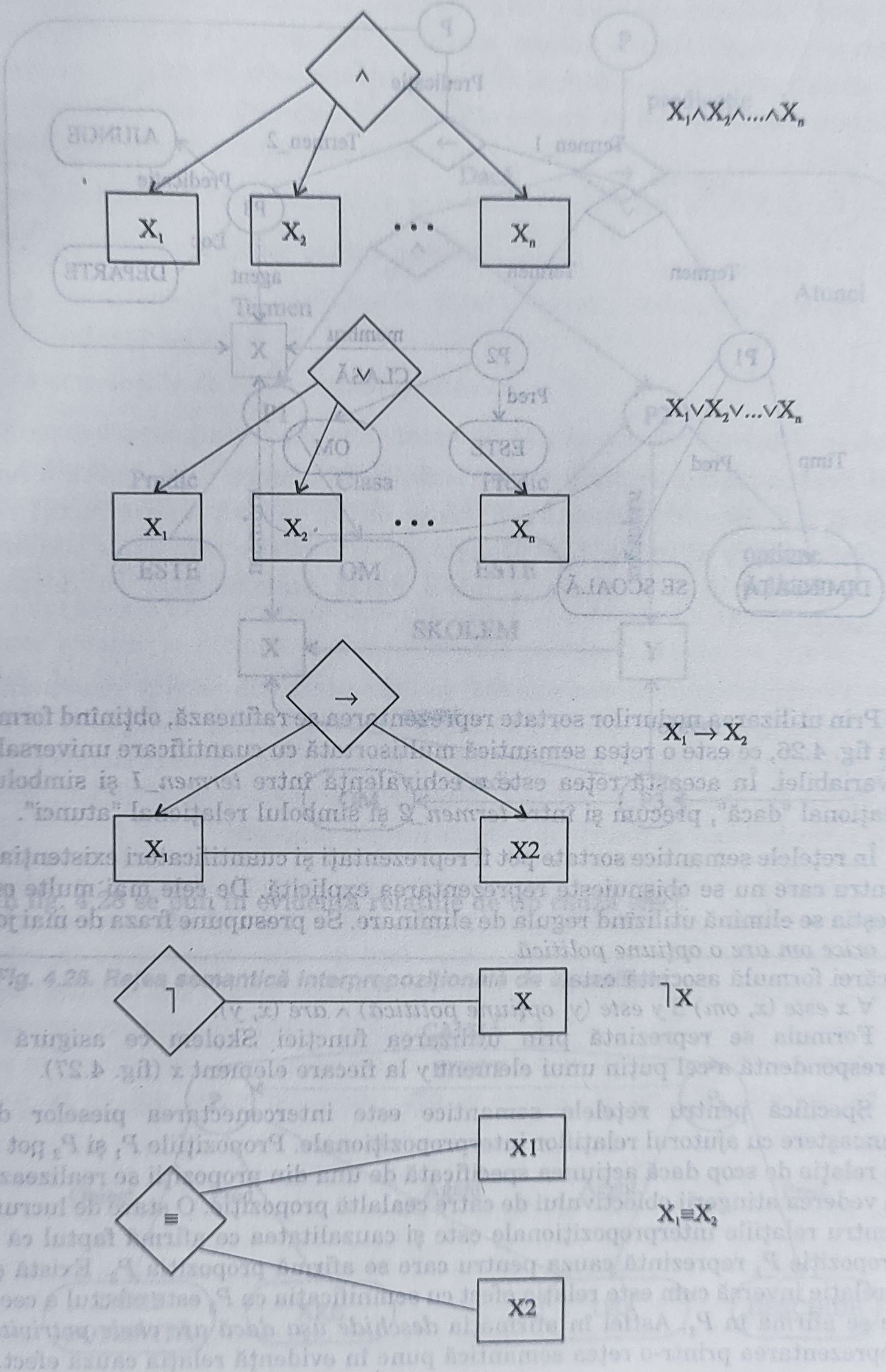


Variabila x ce nu a fost specificată în exemplul inițial dar a fost adăugată ca piesă auxiliară are o participare subînțeleasă la propoziția P_3 . Cu aceste observații poate fi surprinsă semnificația reală a afirmației referitoare, nu orice fel de x , ci numai la acel x ce aparține clasei oamenilor și face ca implicația $P_1 \rightarrow P_2$ să fie adevărată. Cu acest considerente exemplul se va reformula astfel:

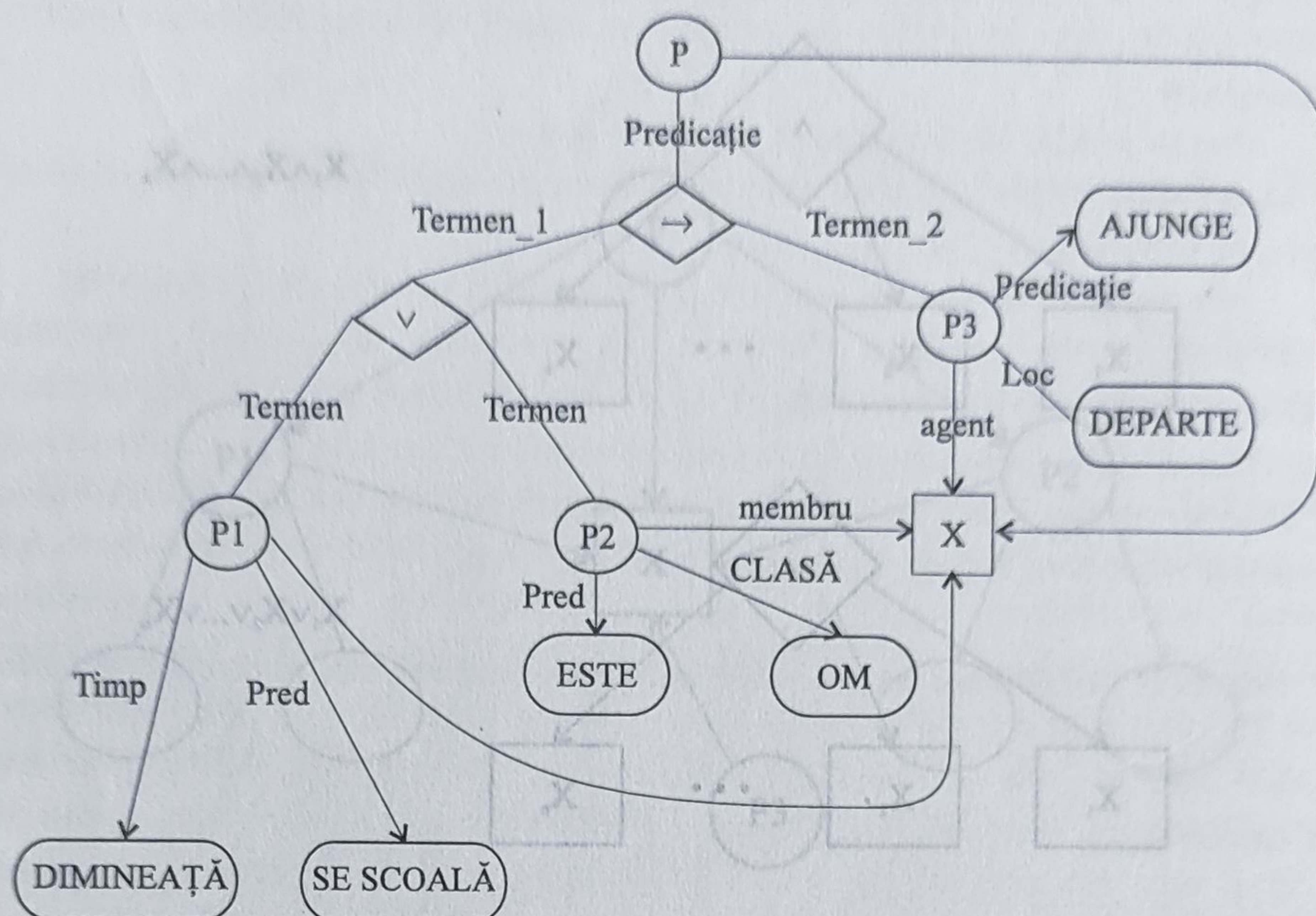
Dacă x este om și x se scoală de dimineață, atunci x ajunge departe.

Reprezentarea frazei într-o rețea semantică necesită un instrument adecvat pentru reprezentarea conectivei ȘI, a conectivei SAU, și a altor conective logice. În acest scop se utilizează sorturi de noduri etichetate drept conective logice, noduri ce sînt în relație cu alte noduri ce reprezintă termeni ai unei formule pe care o determină. Nodurile etichetate ca sorturi de conective logice sînt reprezentate prin romburi și sînt prezentate în fig. 4.25.

▼ Fig. 4.25. Reprezentarea nodurilor specializate



▼ Fig. 4.26. Rețea sortată cu cuantificare universală



Prin utilizarea nodurilor sortate reprezentarea se rafinează, obținînd forma din fig. 4.26, ce este o rețea semantică multisortată cu cuantificare universală a variabilei. În această rețea este o echivalență între *termen_1* și simbolul relațional "dacă", precum și între *termen_2* și simbolul relațional "atunci".

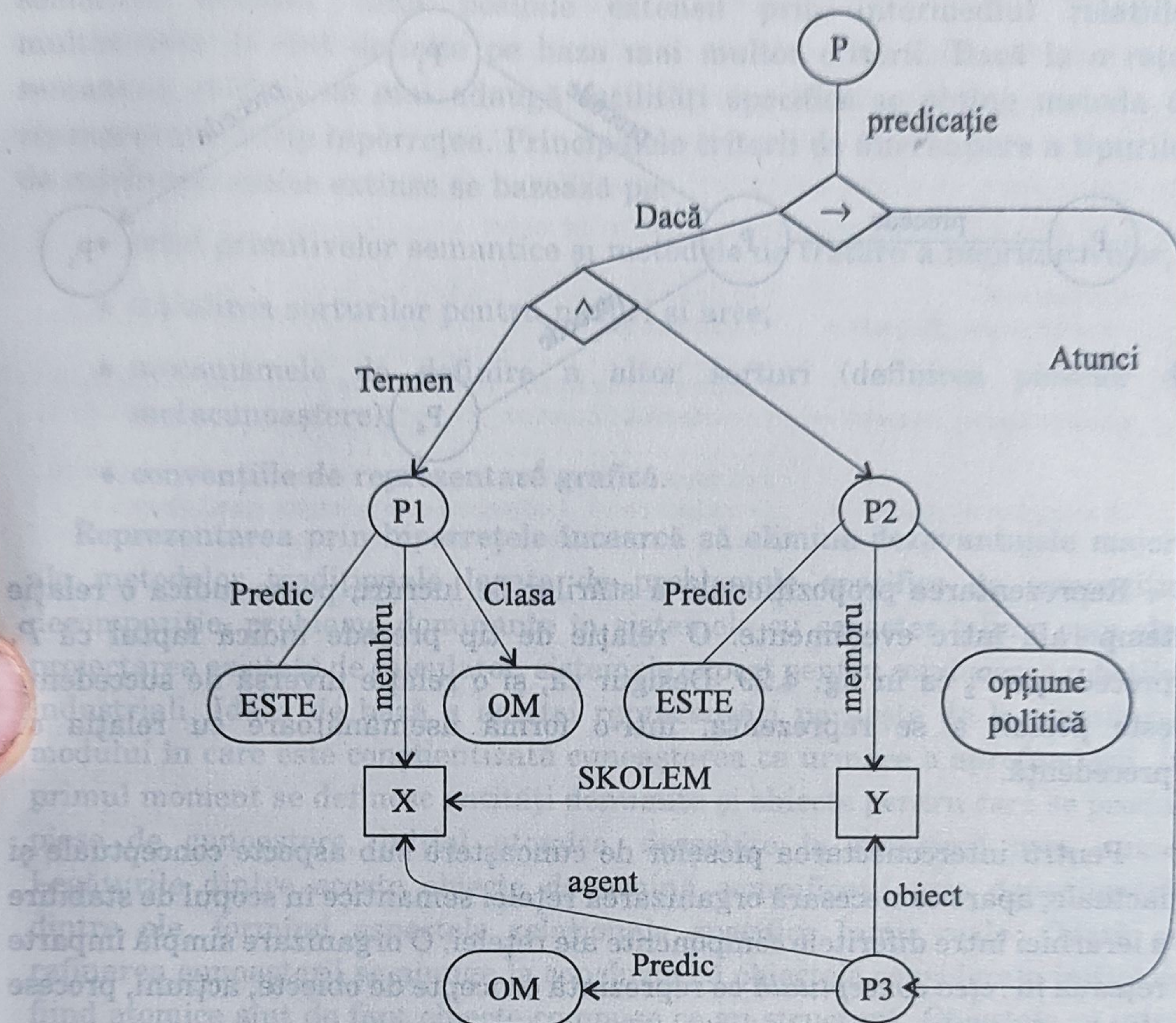
În rețelele semantice sortate pot fi reprezentați și cuantificatori existențiali pentru care nu se obișnuiește reprezentarea explicită. De cele mai multe ori aceștia se elimină utilizînd regula de eliminare. Se presupune fraza de mai jos
orice om are o opțiune politică
 a cărei formulă asociată este

$$\forall x \text{ este } (x, \text{om}) \exists y \text{ este } (y, \text{opțiune politică}) \wedge \text{are } (x, y).$$

Formula se reprezintă prin utilizarea funcției Skolem ce asigură o corespondență a cel puțin unui element y la fiecare element x (fig. 4.27).

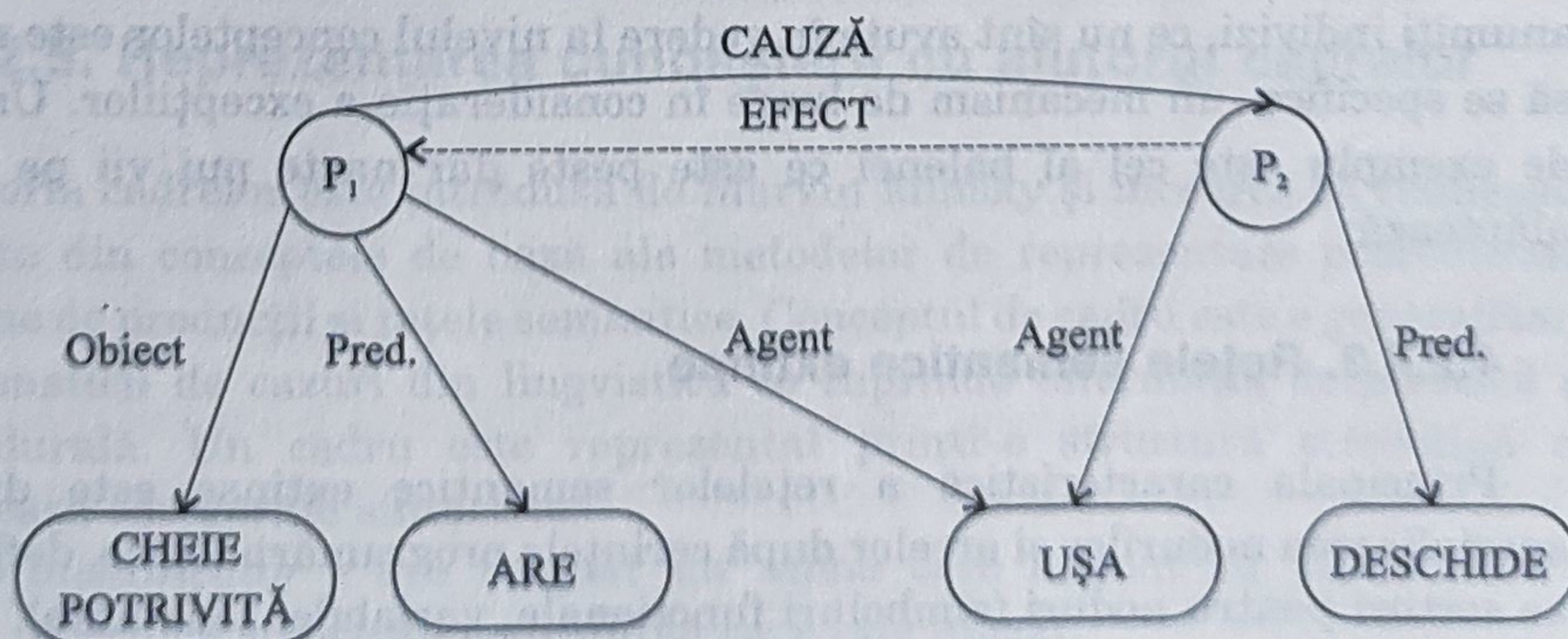
Specifică pentru rețelele semantice este interconectarea pieselor de cunoaștere cu ajutorul relațiilor interpropoziționale. Propozițiile P_1 și P_2 pot fi în relație de scop dacă acțiunea specificată de una din propoziții se realizează în vederea atingerii obiectivului de către cealaltă propoziție. O stare de lucruri pentru relațiile interpropoziționale este și cauzalitatea ce afirmă faptul că o propoziție P_1 reprezintă cauza pentru care se afirmă propoziția P_2 . Există și o relație inversă cum este relația efect cu semnificația că P_2 este efectul a ceea ce se afirmă în P_1 . Astfel în afirmația *deschide ușa dacă are cheie potrivită* reprezentarea printr-o rețea semantică pune în evidență relația cauza efect.

▼ Fig. 4.27. Rețea sortată cu cuantificare existențială

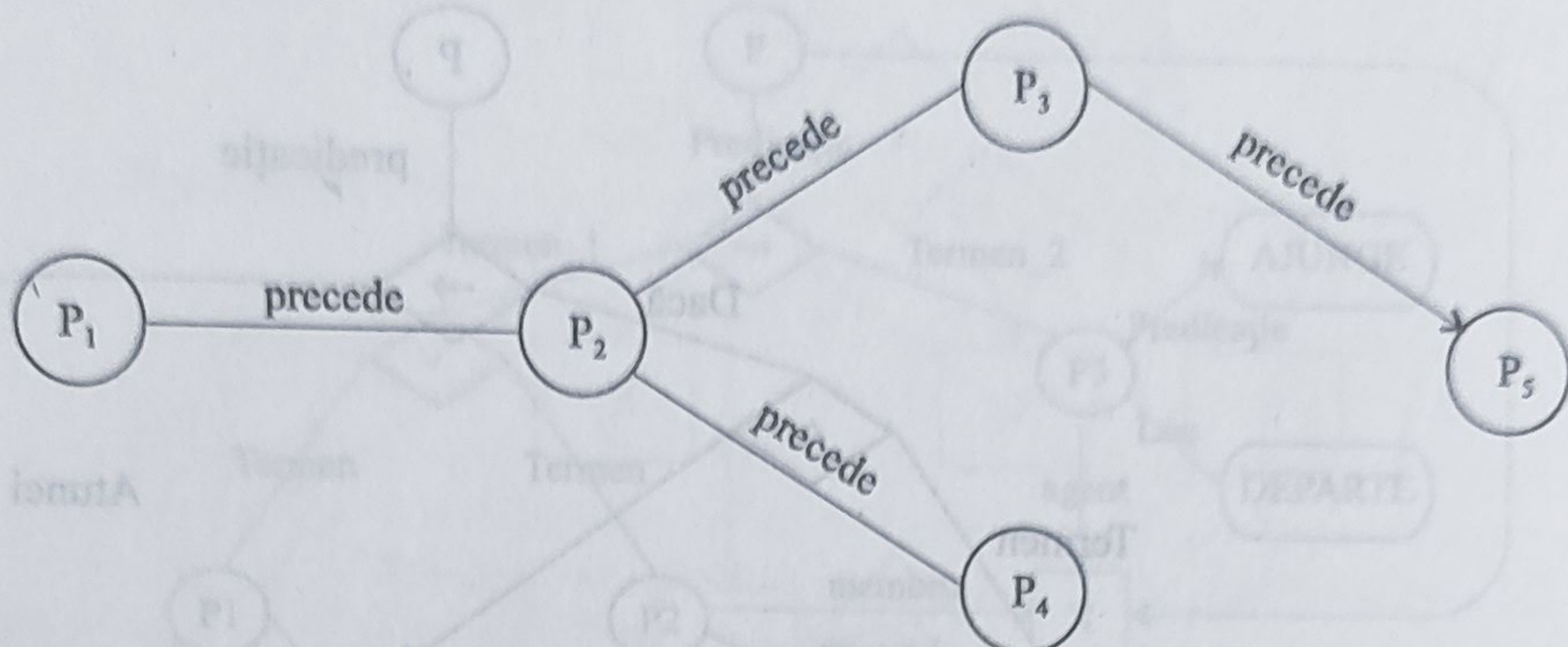


În fig. 4.28 se pun în evidență relațiile de tip cauză efect.

▼ Fig. 4.28. Rețea semantică interpropozițională de cauzalitate



▼ Fig. 4.29. Dependența temporală



Reprezentarea propozițională a stărilor de lucruri, poate indica o relație temporală între evenimente. O relație de tip precede indică faptul că P_1 precede pe P_2 ca în fig. 4.29. Desigur că, și o relație inversă de succedență este posibil a se reprezenta, într-o formă asemănătoare cu relația de precedență.

Pentru interconectarea pieselor de cunoaștere sub aspecte conceptuale și factuale, apare ca necesară organizarea rețelei semantice în scopul de stabilire a ierarhiei între diferitele componente ale rețelei. O organizare simplă împarte rețeaua în *rețea conceptuală* ce reprezintă concepte de obiecte, acțiuni, procese și *rețea factuală* ce descrie instanțele acestor concepte.

Relația taxonomică între concepte și instanțe face ca acestea să moștenească proprietățile conceptelor. Dacă între componentele rețelei factuale apar și relații suplimentare față de cele reprezentate în rețeaua conceptuală ce descriu stări de lucruri particulare la care sînt implicați numai anumiți indivizi, ce nu sînt avute în vedere la nivelul conceptelor este necesar să se specifice un mecanism de luare în considerație a excepțiilor. Un astfel de exemplu este cel al balenei ce este peste dar naște pui vii pe care îi alăptează.

4.2.4.3. Rețele semantice extinse

Principala caracteristică a rețelelor semantice extinse este dată de specializarea nodurilor și arcelor după cerințele programării logice, definindu-se sorturi pentru noduri (simboluri funcționale, variabile, constante), sorturi

pentru arce (relații condiționale și conclusive). Corespondența între funcția semantică și relația folosită se face pe baza oricărei metode pentru rețelele semantice sortate, fiind posibile extensii prin intermediul relațiilor multisortate ce sînt definite pe baza mai multor criterii. Dacă la o rețea semantică extinsă se mai adaugă facilități specifice se obține metoda de reprezentare de tip hiperrețea. Principalele criterii de diferențiere a tipurilor de rețele semantice extinse se bazează pe:

- ♦ setul primitivelor semantice și metodele de tratare a neprimitivelor;
- ♦ stabilirea sorturilor pentru noduri și arce;
- ♦ mecanismele de definire a altor sorturi (definirea pieselor de metacunoaștere);
- ♦ convențiile de reprezentare grafică.

Reprezentarea prin hiperrețele încearcă să elimine dezavantajele majore ale metodelor tradiționale legate de problemele specifice de compoziție, decompoziție, probleme dominante în sistemele cu caracter tehnic cum sînt proiectarea asistată de calculator, sistemele expert pentru conducerea roboților industriali. Ideea de bază a acestei reprezentări pornește de la asimilarea modului în care este conștientizată cunoașterea ca urmare a aprofundării. În primul moment se definesc entități denumite și obiecte pentru care se produc piese de cunoaștere, inițial atomice, deosebite în principal prin nume. Legăturile dintre aceste obiecte determină semnificații prin dependentele dintre ele, formînd aspectele relaționale specifice lumii reale. Odată cu rafinarea cunoașterii se ajunge la concluzia că obiectele considerate inițial ca fiind atomice sînt de fapt obiecte compuse ce au structură. Obiectele ce intră în componența acestora sînt considerate noi obiecte atomice cu termenii cărora se încearcă descrierea structurii noilor obiecte compuse, componența și legăturile acestora. În final de la o relație presupusă binară se obține un graf de aritate superioară.

4.2.5. Reprezentarea cunoașterii cu ajutorul cadrelor

Teoria cadrelor este introdusă de Marvin Minsky și încearcă să reunească o parte din conceptele de bază ale metodelor de reprezentare procedurală, sisteme de producții și rețele semantice. Conceptul de cadru este o generalizare a gramaticii de cazuri din lingvistică ce cuprinde informația declarativă și procedurală. Un cadru este reprezentat printr-o structură stereotipă ce utilizează categoriile sintactice:

- ♦ *identificator* - are asignat un nume care asociat cu alte descrieri specifică structura cadrului la care este atașat;

♦ *forma* - categorii de caracteristici la care se asignează simboluri relaționale specifice conceptului;

♦ *fațeta* - sînt reprezentate de perechi *simbol-valoare* cu care se descriu obiectele din relațiile specificate de formele reprezentării.

Cadrul este format dintr-o structură arborescentă de locuri în care piesa de cunoaștere va plasa simbolurile purtătoare de semnificație. Un formalism metalingvistic de reprezentare a cadrelor este:

```
<cadru> =          (<identificator_cadru> <descriere_formă>)
<descriere_formă>= ((<identificator_formă> <descriere_fațetă>),
.....
(<identificator_formă> <descriere_fațetă>)).
<descriere_fațetă>= ((<identificator_fațetă> <descriere_dată>)),
.....
(<identificator_fațetă> <descriere_dată>)).
<descriere_dată> = (<valoare>) (<descriere_date> <valoare>)
<valoare> = (<date> <date> <eticheta> <mesaje> <comentariu>).
```

Implementările cunoașterii utilizînd conceptul de cadru diferă prin detaliile specifice acestora, așa că anumite implementări orientate spre limbajul LISP dau pentru neterminalele <identificator_cadru>, <identificator_formă>, <identificator_fațetă> reguli de definire diferențiate prin nivelul de plasare a simbolului în structura de listă. Se folosește ca prim simbol identificatorul cadrului, al doilea identificatorul formei, al treilea cel al fațetei, descrierile nestructurate avînd asociate cuvinte cheie pentru cadre, forme, fațete. Simbolurile pot reprezenta proprietăți ale obiectelor sau relațiilor, nume de alte cadre, identificatori de proceduri atașate.

Operația de populare în locurile structurii a simbolurilor și valorilor poartă denumirea de umplere sau "fill-in". Se disting în reprezentarea prin cadre o serie de primitive semantice ce fac să plaseze cunoașterea în conceptul mai larg oferit de structura cadrului cum sînt:

- ♦ forme utilizate la descrierea piesei de cunoaștere;
- ♦ forme ce exprimă reprezentări legate de utilizarea cadrului;
- ♦ forme corespunzînd prezumpțiilor considerate adevărate;
- ♦ forme ce indică reorientarea continuării în situația eșecului.

Relațiile aferente utilizării formelor, relații ce caracterizează piesa de cunoaștere și legăturile sale cu universul problemei se pot împărți după tip în:

- ♦ *generalizare* - specifică concepte cu definiții mai puțin restrictive;
- ♦ *specializare* - specifică concepte cu descrieri ce satisfac o condiție dată;
- ♦ *apartenența* - indică clasa din care face parte piesa de cunoaștere și prin care se moștenesc proprietățile;

- ♦ *compoziția* - indică obiectele ce intră în alcătuirea obiectului;
- ♦ *proprietățile* - reprezintă proprietățile specifice ale obiectului reprezentat.

Se dă mai jos reprezentarea cadru pentru o universitate cu profil tehnic care folosește forme cadru specifice de descriere

cadru: universitate tehnică

forma: generalizare

fațeta: institut de învățământ superior

forma: specializare

fațeta: pregătire tehnică

forma: apartenență

fațeta: universitate

forma: adresa

fațeta: strada

fațeta: număr

fațeta: cod poștal

fațeta: oraș

fațeta: țara

forma: compoziție

fațeta: conducere

valoarea: rector

valoarea: prorector științific

valoarea: prorector administrativ

valoarea: secretariat științific

fațeta: compartimente

valoarea: electric

valoarea: mecanic

valoarea: chimic

forma: proprietăți

fațeta: pregătire

valoarea: curs de zi

valoarea: curs seral

valoarea: curs postuniversitar.

Piese de cunoaștere satisfac definiția cadrelor dată mai înainte. Reglementările privind utilizarea cadrului au în vedere relațiile acestuia cu descrieri de operații, acțiuni, procese. Prelucrările aferente reprezentării prin cadre privesc operațiile ce se efectuează asupra pieselor de cunoaștere cît și operații ce au ca rezultate obiecte printre care se află piese de cunoaștere.

Această metodă pornește de la reprezentarea oferită de LISP, la care cunoașterea este văzută ca perechi atribut_valoare, aspectele relaționale fiind mai puțin relevante decît la reprezentarea prin rețele semantice. Raționamentul implică acțiune, motiv pentru care este necesară reprezentarea cunoașterii despre acțiune. Acțiunea se referă la descrierea acțiunii propriu-

zise formată din cadre ale căror forme reprezintă pași ai acțiunii în care fațetele sînt caracteristicile pașilor (operanzi, cazuri, obiecte), descrierea de acțiuni implicite prin proceduri la care obiectul va fi plasat în modelul unei situații problematice. O procedură atașată simbolului dintr-un cadru se numește *procedură de comutare* dacă simbolul reprezintă un nume de clasă și procedura este invocată pentru orice element aparținînd clasei, atunci cînd el substituie în procesul de interpretare simbolul clasei în cadrul considerat. Un exemplu tipic este legat de procedurile de umplere a valorilor asociate formelor într-o reprezentare prin cadre, în care procedura asociată este de forma:

cadru: individ

forma: nume

fațeta: comutație

valoare: umple nume (nume).

Procedura asigură umplerea cu simboluri și valori a formei nume ce reprezintă un cadru referit procedural. Se creează o procedură identică cu apelarea de subrutine din mai multe cadre sau din mai multe puncte ale aceluiași cadru. Dacă procedura este atașată la un simbol unic atunci se încorporează în corpul cadrului imediat următor simbolului de invocare. Alte proceduri folosesc atașarea implicită și sînt invocate nu ca urmare a specificării în cadrele unde figurează simboluri la care se atașează proceduri ci ca urmare a apariției unei situații specifice. În acest caz procedura se cuplează la apariția situației, motiv pentru care este numită și *demon*. Supravegherea situației specifice de către demon se realizează prin:

- ♦ apariția unui fapt în baza de cunoștințe ce corespunde unui patern;
- ♦ apariția în procesul de interpretare a unei condiții de invocare pentru un patern dat.

Instalarea demonului în LISP se poate face printr-o funcție definită de programator avînd o structură de tipul

`<demon> = (def_demon<nume><tip><patern><corp>)`

4.2.5.1. Reprezentarea acțiunii în cadre

Pentru reprezentarea acțiunilor se poate asocia unei forme simbol aparținînd categoriei semantice a programelor cu fațete ce reprezintă secvențe de operații, secvențe de stări sau secvențe de evenimente. Astfel cadrul conține pe lîngă componente declarative și o componentă cu caracter procedural. Conceptul de scenariu face posibilă sistematizarea modului de folosire a substructurilor cadrului în vederea folosirii interpretative a acestuia. Partea procedurală are o structură ce reproduce conceptul de program ce indică

acțiunea normală, acțiunea în cazul lipsei unui obiect, acțiunea în caz de eroare. Partea declarativă, conține informații despre inițierea și completarea acțiunii astfel încât să fie anticipate toate aspectele sale funcționale. Fie de exemplu următorul scenariu:

scenariu: curs

agent: cadru didactic

loc_desfășurare: sala de curs

timp: 25 aprilie 1991, ora 8

program:

eveniment_1: intrarea studenților în sală

eveniment_2: intrarea profesorului în sală

eveniment_3: desfășurare prima oră de curs

eveniment_4: pauză

eveniment_5: intrarea în sală

eveniment_6: desfășurarea celei de a doua oră de curs

eveniment_7: profesorul părăsește sala

eveniment_8: studenții părăsesc sala

început: eveniment_1

obiectiv: eveniment_7.

Acest scenariu poate fi reprezentat ca o succesiune de forme ce conțin simboluri cu pointeri la cadre de detaliu. În exemplul de mai sus profesorul este reprezentat printr-un cadru ce are în compoziție forme ce precizează entitatea. În procesul de interpretare faptele urmăresc succesiunea de evenimente, ce se desfășoară după scenariul specificat fără a exista evenimente marcate cu probabilități de producere. Se poate stabili că dacă evenimentele 1, 2 și 5 s-au produs atunci și evenimentele 3 și 6 se vor produce cu siguranță. Renunțând la imaginea de perfecțiune se poate determina un program de evenimente ce specifică alternativele posibile în caz de eșec. Demonii instalați au rolul de a trata cazuri cum este cel profesor bolnav, și va indica trecerea la evenimentul_8.

4.2.5.2. Introducerea regulilor în cadre

O facilitate puternică este cea a introducerii în cadre a reprezentării regulilor. Transpunerea unei reguli în cadre pornește de la obiectivele:

- ♦ regula conține două acțiuni distincte deci două forme procedurale diferite;
- ♦ prima formă cea de evaluare a condițiilor are ca efect autorizarea execuției celei de a doua forme procedurale sau ieșirea din regulă la eșec;

- ♦ a doua formă determină corpul propriu-zis al acțiunii ce se execută când condiția este îndeplinită;
- ♦ contextul operațional este reprezentat de celelalte forme cu caracter declarativ.

Se consideră în exemplul de mai jos regula:

IF: rezultatul testului de citire-scriere este eronat, bitul 2 blocat în 1

(and) se manifestă la circuitul M1

(and) circuitul M2 funcționează corect

THEN: circuitul M1 este defect

(and) reparare prin înlocuire

se va reprezenta prin cadre astfel:

Cadru R1

forma: condiție

fațeta: rezultat test citire-scriere eronat, bit 2 blocat în 1

fațeta: se manifestă la circuitul M1

fațeta: M2 funcționează corect

forma: acțiune

fațeta: defect

valoare: M1

fațeta: reparație

valoare: înlocuire

Structura implicit ierarhizată a conceptului de cadru face posibilă organizarea sub formă de arbori de clasificare. Structura arborescentă derivă din cadre prototip ale claselor și subclasselor.

Un sistem rezolutiv este format din totalitatea componentelor unui sistem de inteligență artificială avînd ca obiectiv rezolvarea de probleme. Prin problemă se înțelege o noțiune fundamentală de psihologie a gîndirii ce se referă la dificultatea de natură cognitivă ce se constituie ca moment inițial al activităților inteligente. Dificultatea pornește de la imposibilitatea asocierii unor răspunsuri la întrebările asociate unei entități necunoscute. Problema formulată spre rezolvare unui sistem de inteligență artificială este un enunț de problemă potențială, confirmarea enunțului unei astfel de probleme externe, reale, depinde de apariția unei situații discordante între obiectivele problemei, faptele din baza de cunoștințe, metodele disponibile pentru corelarea cu obiectivele problemei. Transpunerea problemei de la subiectul uman la sistemul de inteligență artificială presupune reformularea acesteia într-un formalism ce reflectă situația problematică actuală creată ca urmare a discordanței între obiectivele specificate în formularea externă și resursele cognitive și rezolutive ale sistemului.

Mecanismul de inferență este inima oricărui sistem expert, care alimentat de baza de cunoștințe construiește raționamentul în mod dinamic decizînd ce reguli sînt declanșate și în ce ordine. Astfel, raționamentele declanșate pentru reprezentarea prin reguli de producție sînt cele în logica formală "*modus ponens*" și "*modus tollens*".

Componentele primitive constituente ale problemei pot fi grupate în:

- ♦ aserțiuni referitoare la obiecte abstracte sau concrete formînd pentru problema dată universul discursului. Acestea alcătuiesc aspectul factual al enunțului cunoscut sub denumirea de date ale problemei;
- ♦ aserțiuni explicite sau implicite referitoare la operațiile permise a fi aplicate obiectelor menționate. Aceste aserțiuni formează aspectul procedural al enunțului, cunoscut și sub denumirea de condiții ale problemei;

- ♦ obiecte, proprietăți a căror existență este o consecință a enunțului formînd ceea ce se numesc "necunoscute ale problemei".

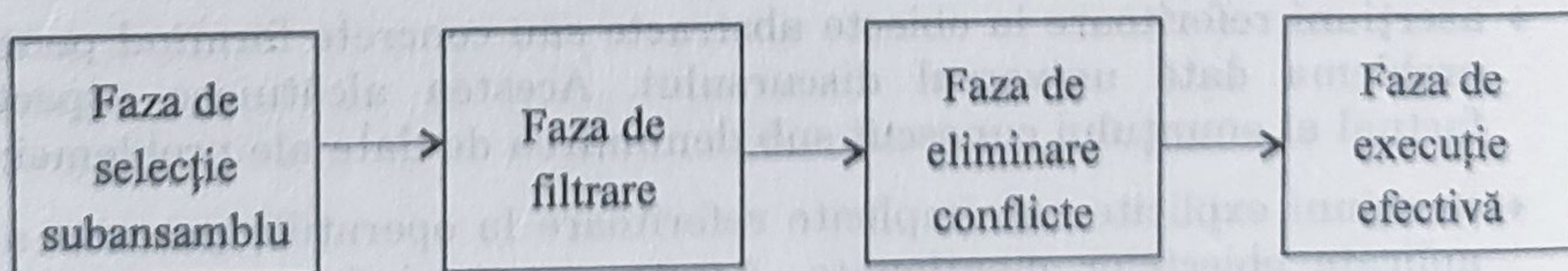
Obiectivul rezolvării oricărei probleme este de a furniza o determinare pentru fiecare din necunoscute. Universul discursului în procesul rezolvării este delimitat mai întîi de datele și condițiile formulate prin enunț. La sistemele inteligente bazate pe cunoașterea universului inițial i se adaugă piese de cunoaștere din baza de cunoștințe ce sînt selectate fie prin referire directă la enunț fie prin inferente logice cu premise formulate în enunț.

5.1. Ciclul de bază al unui mecanism de inferență

Indiferent de modul de raționare utilizat ciclul de bază al unui mecanism de inferență cuprinde patru faze (fig. 5.1) și anume:

- ♦ faza de *selecție* a unui subansamblu al bazei de fapte și reguli ce merită atenția față de restul bazei. Selecția oferă o economie de timp considerabilă pentru fazele următoare. Desigur că sistemul de inferență nu trebuie să privilegieze un grup de reguli față de altele;
- ♦ faza de *filtrare* ce are ca efect comparația între partea de premisă a regulii considerate și faptele bazei de fapte pentru determinarea regulilor aplicabile. Aplicarea după faza de selecție a fazei de filtrare ajută la reducerea substanțială a ansamblului regulilor ce sînt filtrate cu elementele din baza de fapte;
- ♦ faza de *rezolvare a conflictelor* are ca obiectiv alegerea acelor reguli ce sînt aplicate efectiv. Problema este rezolvată printr-o strategie ce poate fi foarte simplă în raport cu contextul sau mai complexă ținînd cont de context în sensul aplicării regulii cît mai promițătoare;
- ♦ faza de *execuție* constă din aplicarea regulilor alese mai înainte, acțiunea constînd în general în adăugarea de noi fapte în baza de fapte. Este posibil ca aplicarea regulii să facă apel la proceduri externe avînd ca efect modificări ale bazei de fapte, formularea de întrebări către utilizator.

▼ Fig. 5.1. Fazele procesului rezolutiv



Lanțurile inferențiale sau rețelele inferențiale între premise și concluzie determină procesul numit și *raționament*, ce se bazează pe faptele furnizate prin enunț și pe piesele de cunoaștere existente în baza de cunoștințe. Atunci când faptele furnizate în enunț și piesele de cunoaștere din baza de cunoștințe sînt suficiente, raționamentul se numește *direct*, întrucît entitățile din rețeaua inferențială aparțin faptelor aferente problemei. Incompletitudinea faptică duce la impas, și au fost studiate și alte tipuri de raționamente cum este de exemplu raționamentul *metaforic* sau raționamentul prin analogie.

Metodele binecunoscute pînă în prezent sînt cele ce fac parte din clasa raționamentelor directe. Metodele ce studiază transformările de la premise la concluzii se numesc și metode de raționare formală și au o importanță deosebită în abstractizarea unor proprietăți ce intervin la raționamentele prin analogie.

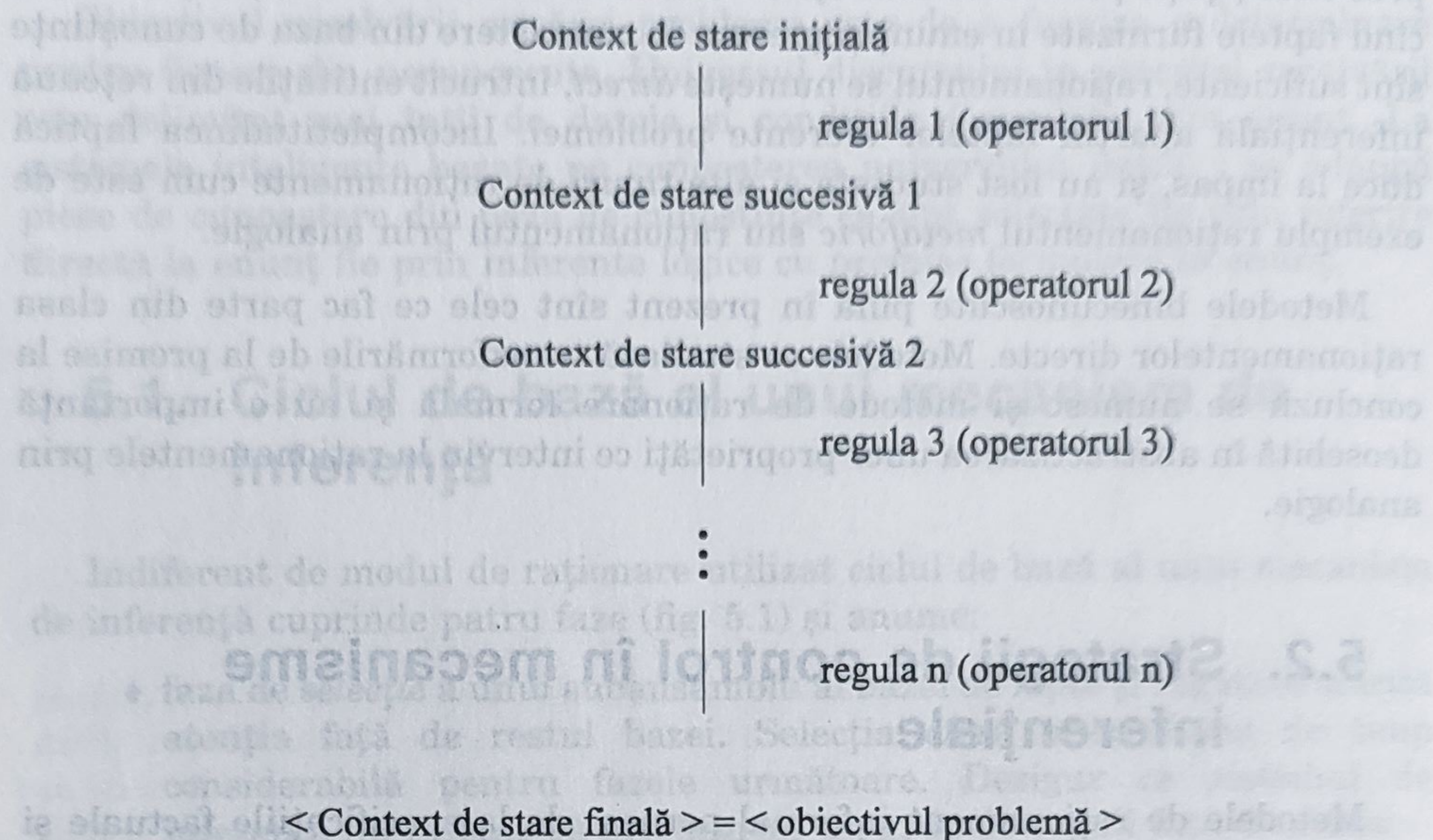
5.2. Strategii de control în mecanisme inferențiale

Metodele de raționament informal pornesc de la specificațiile factuale și procedurale cuprinse în enunț. Aceste metode de raționament surprind interpretări structurale, semnificații de natura relațională, proprietăți primitive, structuri ierarhice bazate pe primitive semantice. Raționamentul informal este necesar să se valideze prin aplicarea regulilor specifice simbolurilor folosite actual. Desfășurarea raționamentelor are loc pe baza unei strategii ce asigură succesiunea inferențelor. Se dau mai jos cîteva strategii de control al raționamentelor.

5.2.1. Strategia de control înainte

Această strategie pornește de la starea inițială de fapte descrisă prin enunț, stare de la care prin aplicarea de raționamente se generează candidați la soluție pînă la obținerea răspunsului corespunzător obiectivului problemei. Regulile ce se utilizează pot fi reguli de inferență ale mecanismelor logice de bază, reguli specifice definite prin enunț referitoare la generarea termenilor, simboluri relaționale, restricții ce reduc spațiul problemei. Operatorii corespunzători algebrei multisortate ce definește problema pot fi examinați prin reguli. Se mai spune în strategiile de control înainte că se pornește de la fapte pentru a ajunge la obiectiv. În consecință se vor selecționa regulile a căror parte de condiție este verificată (faza de selecție și filtrare). Pentru faza de rezolvare a conflictelor din ansamblul regulilor selectate se vor alege acelea ce au prioritate maximă. Ca urmare, se va obține o strategie generală de control similară cu cea prezentată în fig. 5.2.

▼ Fig. 5.2. Strategia de control înainte



Aplicarea regulilor determină date și fapte noi. Se pot obține astfel la fiecare pas derivări valabile rezultate ca urmare a aplicării mai multor reguli, dar numai una este valabilă în sensul că aparține lanțului ce conduce la soluție. La acest moment nu se cunoaște această regulă și ca urmare se vor aplica toate regulile ce sînt posibil aplicabile obținînd o mulțime de stări caracterizate prin date și fapte. Succesiunea sugerează un arbore ale cărui noduri sînt stări în spațiul stărilor, soluția problemei fiind dată printr-o procedură de căutare în spațiul stărilor. Ținînd cont de caracteristicile metodei ce se bazează pe strategia de control înainte, metoda mai este numită și productivă, iar strategia se mai numește și *strategie de căutare dirijată prin date sau de jos în sus*. Procesul de căutare continuă pînă cînd mulțimea regulilor aplicabile devine vidă. Un astfel de proces poate fi transpus în pseudocod astfel:

```

Stabilirea faptelor inițiale
begin
  faza de filtrare (determinarea mulțimii regulilor aplicabile)
  while (ansamblul regulilor aplicabile nu este vid și problema nu
    a fost rezolvată)
    execută faza de lucru (rezolvarea conflictelor)
      aplicarea regulii alese
      modificarea ansamblului regulilor aplicabile
    end executa
  end while
end

```

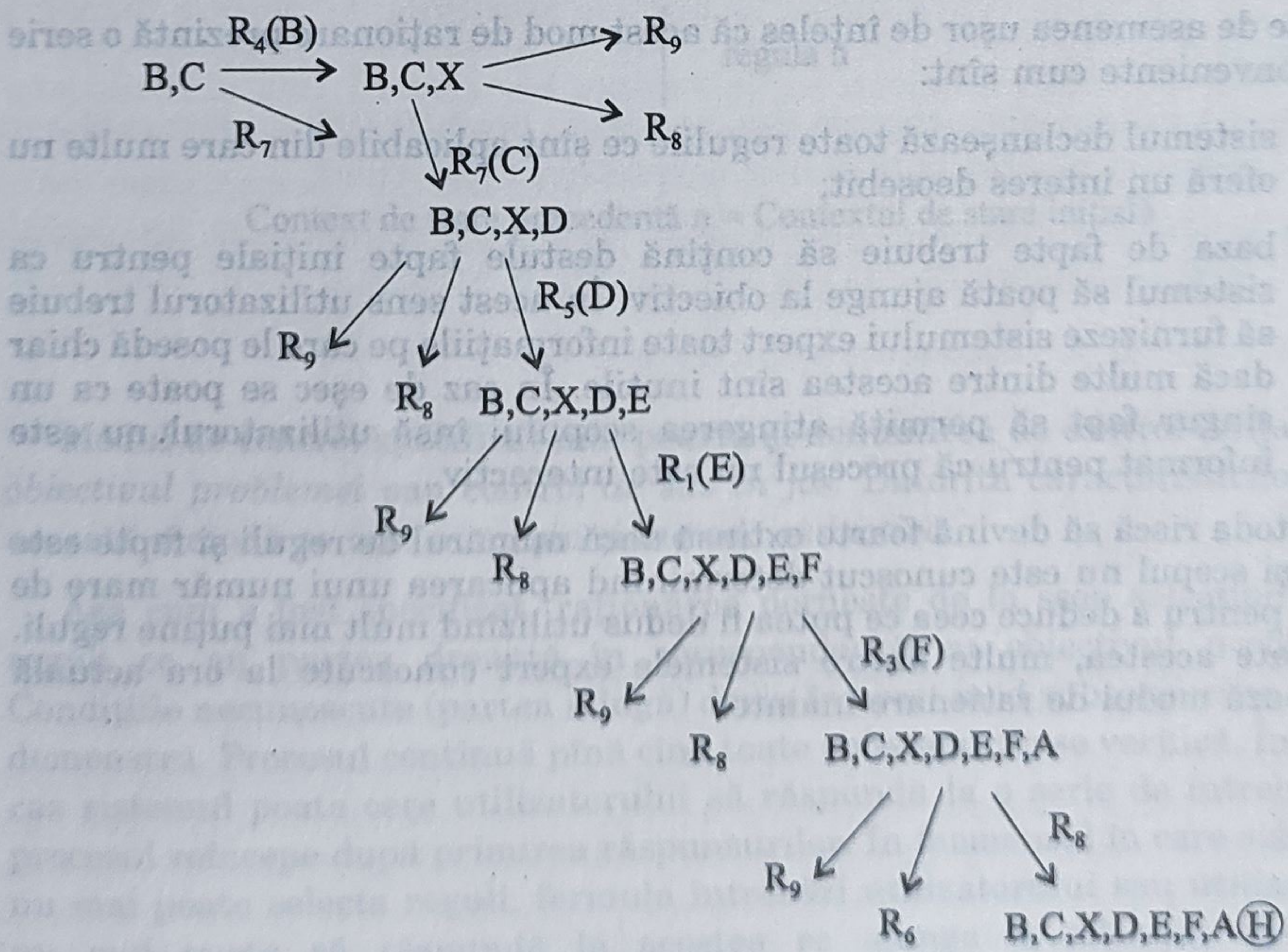

Se poate spune că eficacitatea mecanismului de inferență constă în relevanța deciziei luate în faza de lucru, fapt ce determină și rapiditatea cu care sistemul ajunge la soluție. Ca efect, rezolvarea unei probleme poate determina un număr foarte mare de cicluri de bază. Este cert că pentru un calculator universal fiecare ciclu de bază determină un număr mare de cicluri de instrucțiune. Viteza de execuție este de ordinul miilor de cicluri de inferență pe secundă la milioane de cicluri de inferență (LIPS Logical Inference Per Second). Pornind de la această observație este justificat interesul de scădere pe cât posibil a numărului de cicluri de inferență.

Se consideră mai jos o bază de cunoștințe reprezentată prin reguli de producție:

R_1 : IF B and D and E	THEN F
R_2 : IF D and G	THEN A
R_3 : IF C and F	THEN A
R_4 : IF B	THEN X
R_5 : IF D	THEN E
R_6 : IF A and X	THEN H
R_7 : IF C	THEN D
R_8 : IF X and C	THEN A
R_9 : IF X and B	THEN D

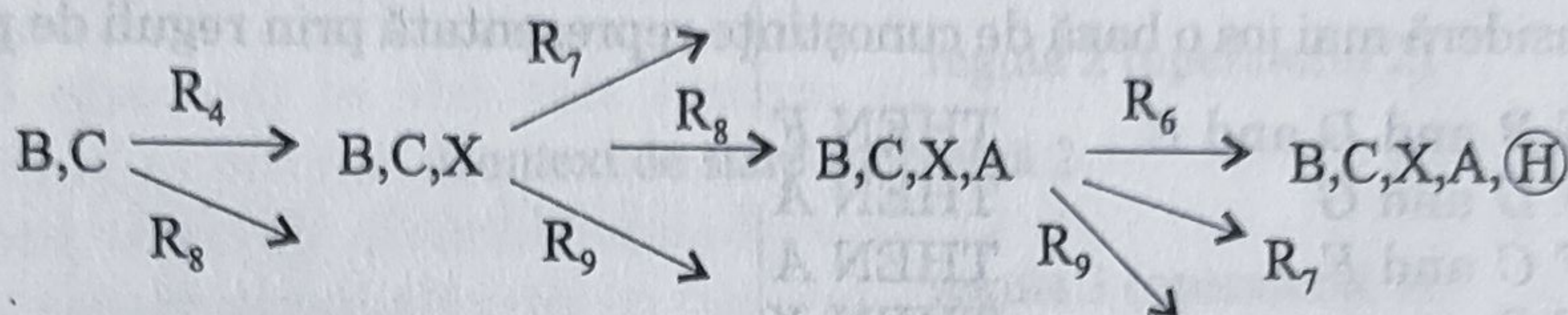
• Dacă se presupune baza de fapte inițială B , C și obiectivul H , construcția arborelui de decizie pornind de la prima regulă prin secvențierea regulilor în ordinea în care apar în baza de cunoștințe se obține următorul arbore (fig. 5.3):

▼ **Fig. 5.3. Aplicarea regulilor în ordinea stocării**



Se consideră la acest moment o strategie prin care se selectează din mulțimea regulilor aplicabile acelea pentru care numărul de condiții din partea de premiză este mai mare. Se observă că regula R_8 este preferabilă față de regula R_7 întrucât are în premisă două condiții față de R_7 care are doar una. Cu această strategie se obține arborele din fig. 5.4.

▼ Fig. 5.4. Selecția regulilor funcție de numărul condițiilor din premisă



Se observă că printr-o astfel de strategie numărul de inferențe se reduce la trei față de șase în cazul anterior. În plus trebuie reținut faptul că o inferență aplicată odată nu se mai aplică întrucât nu servește la nimic să se deducă un fapt ce a fost deja dedus. Eficacitatea ajungerii la scop este dată de numărul de inferențe necesare, număr de inferențe ce depinde în mare măsură de faza de eliminare conflicte. Strategia utilizată în fig. 5.3 deși este foarte simplă are dezavantajul de a fi puternic dependentă de ordinea regulilor din baza de cunoștințe. Chiar dacă strategia utilizată în fig. 5.4 este mai complicată din punctul de vedere al implementării se justifică prin rezultatele obținute, mai ales prin rapiditate.

Este de asemenea ușor de înțeles că acest mod de raționare prezintă o serie de inconveniente cum sînt:

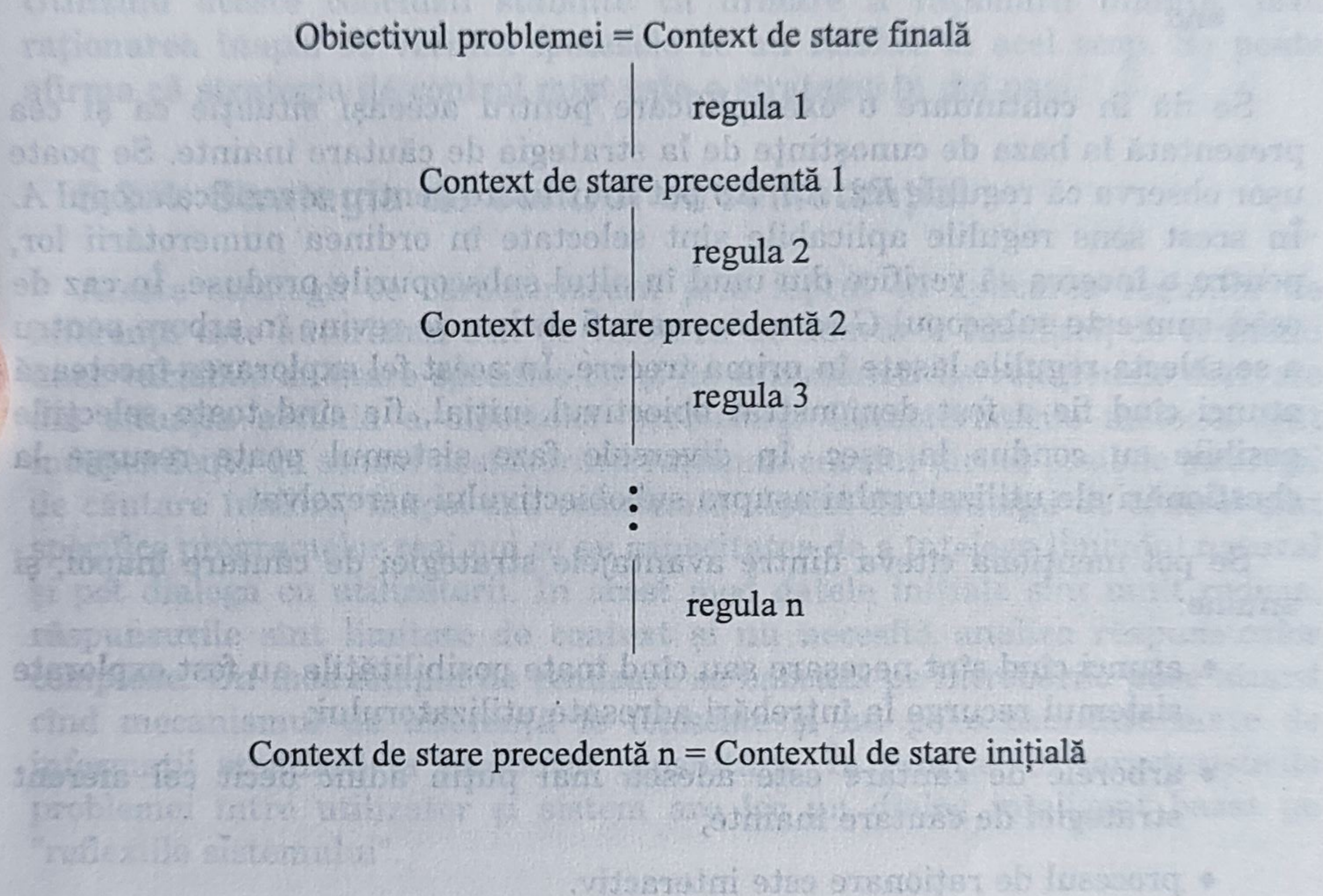
- ♦ sistemul declanșează toate regulile ce sînt aplicabile din care multe nu oferă un interes deosebit;
- ♦ baza de fapte trebuie să conțină destule fapte inițiale pentru ca sistemul să poată ajunge la obiectiv. În acest sens utilizatorul trebuie să furnizeze sistemului expert toate informațiile pe care le posedă chiar dacă multe dintre acestea sînt inutile. În caz de eșec se poate ca un singur fapt să permită atingerea scopului însă utilizatorul nu este informat pentru că procesul nu este interactiv.

Metoda riscă să devină foarte extinsă dacă numărul de reguli și fapte este mare și scopul nu este cunoscut determinînd aplicarea unui număr mare de reguli pentru a deduce ceea ce putea fi dedus utilizînd mult mai puține reguli. Cu toate acestea, multe dintre sistemele expert cunoscute la ora actuală utilizează modul de raționare înainte.

5.2.2. Strategia de control înapoi

Această strategie de control pornește de la obiectivul problemei care prin aplicarea regulilor de descompunere se transformă în subprobleme de complexitate mai mică. Structura generală a inferențierii utilizând metoda de control înapoi este ilustrată în fig. 5.5.

▼ Fig. 5.5. Strategia de raționare înapoi



Modul de control specificat mai poartă și denumirea de control dirijat prin *obiectivul problemei* sau control *de sus în jos*. Datorită caracteristicilor sale această metodă se mai numește și *metodă reductivă*.

Așa cum a fost specificat, raționarea pornește de la scop selectând acele reguli ce au partea dreaptă în corespondență cu obiectivul problemei. Condițiile necunoscute (partea stângă) devin în acest mod subscopuri ce se vor demonstra. Procesul continuă până când toate subscopurile se verifică. În acest caz sistemul poate cere utilizatorului să răspundă la o serie de întrebări și procesul reîncepe după primirea răspunsurilor. În momentul în care sistemul nu mai poate selecta reguli, formula întrebări utilizatorului sau utilizatorul nu mai poate să răspundă la acestea se ajunge la situația de eșec. Transpunerea algoritmului în pseudocod este ilustrată mai jos.


```

begin
  faza de filtrare
  IF ansamblul regulilor selectate este vid
  THEN întrebări adresate utilizatorului
  ELSE
    while problema nu este rezolvată și mai sînt reguli
      selecționate
        executa
          faza de lucru
          adaugă subscopul la partea dreaptă a regulii
          selecționate
          IF subscopul nu este soluționat reluare
          ELSE editează soluția și STOP
        end while
      end if
    end
  end

```

Se dă în continuare o exemplificare pentru aceeași situație ca și cea prezentată la baza de cunoștințe de la strategia de căutare înainte. Se poate ușor observa că regulile R_2 , R_3 , R_8 pot fi utilizate pentru a verifica scopul A . În acest sens regulile aplicabile sînt selectate în ordinea numerotării lor, pentru a încerca să verifice din unul în altul subscopurile produse. În caz de eșec, cum este subscopul G care nu poate fi dedus, se revine în arbore pentru a se selecta regulile lăsate în prima trecere. În acest fel explorarea încetează atunci cînd fie a fost demonstrat obiectivul inițial, fie cînd toate selecțiile posibile au condus la eșec. În diversele faze sistemul poate recurge la chestionări ale utilizatorului asupra subobiectivului nerezolvat.

Se pot menționa cîteva dintre avantajele strategiei de căutare înapoi, și anume:

- ♦ atunci cînd sînt necesare sau cînd toate posibilitățile au fost explorate sistemul recurge la întrebări adresate utilizatorului;
- ♦ arborele de căutare este adesea mai puțin adînc decît cel aferent strategiei de căutare înainte;
- ♦ procesul de raționare este interactiv.

Pe lîngă avantajele menționate mai sus strategia de control înapoi prezintă și un dezavantaj major legat de faptul că există pericolul buclării, în sensul că pentru a demonstra A trebuie demonstrat B iar pentru demonstrarea lui B trebuie demonstrat A .

5.2.3. Strategia de control combinat

Aceste strategii de control sînt caracterizate de faptul că utilizează metode reductive pentru descompunerea problemei în subprobleme care apoi sînt rezolvate prin metode productive. Rezultatul aplicării parțiale a unei metode reductive este descompunerea acesteia în subprobleme ce pot fi tratate în continuare fie prin aplicarea metodelor reductive fie prin schimbarea metodei de tratare și aplicarea, de exemplu, a metodelor productive. Succesiunea

inversă nu este posibilă întrucât metodele productive generează de la primul pas candidați la soluție ce nu reprezintă subprobleme fapt pentru care odată selectată metoda productivă nu poate fi părăsită pentru a continua rezolvarea prin alte metode. Deci odată generată o metodă productivă aceasta va fi utilizată pînă la obținerea soluției finale.

Utilizarea combinată a celor două metode duce la o simplificare a problemei și are ca efect o reducere a spațiului stărilor pentru subproblemele componente cu efecte favorabile în privința necesarului de resurse pentru rezolvare. Cu aceste considerente este clar faptul că nu trebuie confundată strategia de inferență mixtă, înainte în alternanță cu strategia de control înapoi cu metoda de raționare mixtă. În general un sistem expert utilizează strategia de control înainte pentru determinarea unor concluzii parțiale. Utilizînd aceste concluzii stabilite ca urmare a raționării înainte, prin raționarea înapoi se verifică ipotezele ce au condus la acel scop. Se poate afirma că strategia de control mixt este o strategie în doi pași.

5.2.4. Strategia de control circumstanțial

Aceste strategii se caracterizează prin faptul că aplicarea regulilor de inferență este autorizată atît de valoarea de adevăr a condiției, de termenii unor variabile de stare specifice cît și de caracteristicile relaționale derivate din situația actuală a obiectelor problemei. Caracteristicile metodei sînt independente de sensul desfășurării raționamentului făcînd posibile strategii de căutare înainte, înapoi sau combinate. Astfel de strategii de control sînt specifice programelor mai noi ce au capacitatea de a înțelege limbajul natural și pot dialoga cu utilizatorii. În acest mod datele inițiale sînt mult reduse, răspunsurile sînt limitate de context și nu necesită analiza răspunsurilor complexe. Un mod simplu de realizare se bazează pe întrebările puse atunci cînd mecanismul de inferență le folosește și nu pe o cantitate mare de informații standard la începutul consultării. În funcție de caracteristicile problemei între utilizator și sistem are loc un dialog inteligent bazat pe "reflexiile sistemului".

5.2.5. Alegerea strategiei de control

În funcție de tipul de raționament ales, construcția mecanismului de inferență necesită două etape delicate. Prima se referă la definirea metodelor de selecție a regulilor candidate. În acest sens se confruntă fiecare regulă cu ansamblul faptelor din baza de fapte sau cu scopul ce se dorește atins. În alți termeni operația se mai numește și filtrare. A doua etapă se referă la alegerea regulii pentru declanșare. Această alegere influențează performanțele sistemului. Dacă un om știe ce cunoștințe utilizează într-o situație dată un sistem expert va trebui să urmeze o strategie. Un mod de control foarte interesant se bazează pe utilizarea metacunoștințelor. Acest nivel de cunoaștere este specific strategiilor de raționare. Cert este faptul că nici o

strategie de control nu este bună în orice situație. Controlul dirijat prin date este foarte popular, ușor de realizat și se recomandă a se utiliza atunci când se dispune de o cantitate mare de informație neștiind nimic despre scopul ce urmează a fi atins. Strategia de control înainte este indicată pentru tratarea cunoștințelor empirice și este inefficientă pentru probleme complexe. Dacă însă unul sau mai multe scopuri trebuiesc atinse sau verificate este mult mai rațional să se utilizeze strategia de control înapoi. Raționarea înapoi este recomandată în cazul informațiilor incomplete sau când se poate angaja un dialog cu utilizatorul.

5.3. Analiza tipurilor de probleme

Capacitatea de rezolvare a problemelor de către un sistem inteligent este apreciată după ușurința cu care dă soluții la probleme ce nu au fost stabilite în prealabil, precizarea lor fiind făcută în momentul încărcării bazei de cunoștințe.

Din punctul de vedere al satisfacerii condițiilor de formulare se pot distinge trei mari categorii de probleme:

- ♦ *probleme bine formulate* - sînt acele probleme ce satisfac condițiile de necesitate și suficiență pentru toate componentele din structură, necunoscutele împreună cu datele alcătuind un model consistent;
- ♦ *probleme incomplet definite* - sînt problemele în a căror formulare se poate pune în evidență lipsa unor date, a condițiilor acestora sau sînt lipsuri în specificarea elementelor văzute ca necunoscute;
- ♦ *probleme greșit formulate* - pentru care pot fi puse în evidență contradicții și inconsistente sau nu pot fi stabilite clar componentele problemei.

Aplicarea strategiilor de rezolvare specifice tipului de problemă se face după o analiză asupra naturii intime a acesteia, în vederea clarificării de detaliu. Problemele bine formulate pot fi clasificate în: probleme de tip interogativ, probleme de tip predicativ, probleme de tip inductiv.

O problemă de tip interogativ este compusă din trei părți:

- *ipoteza* (alcătuită din date);
- *procesul* la care sînt supuse datele;
- *rezultatul* ce se obține în urma aplicării procesului.

O formalizare completă a problemei de tip interogativ presupune definirea completă a două din cele trei elemente, cea de a treia fiind necunoscută. Dacă se notează cu I ipoteza, cu P procesul, cu R rezultatul și cu X necunoscuta,

funcție de poziția necunoscutei în componentele problemei se deosebesc probleme interogative de tip *IPX*, *IXR* și *XPR*. În situația în care toate componentele menționate sînt cunoscute enunțul problemei este de tip *IPR* și aceasta reprezintă un fapt. Se poate afirma că atunci cînd unul din componentele unui fapt este necunoscut acesta devine o problemă.

O analiză de detaliu arată că problemele interogative de tip *IPX* au enunțul "care este rezultatul x al aplicării procesului P asupra obiectelor din ipoteza I ?" și sînt de fapt problemele clasice de prelucrare a datelor. Desigur că enunțul problemei este cunoscut sau completat cu informații preluate din baza de cunoștințe. Problema interogativă de tip *IXR* corespunde enunțului "ce proces X trebuie aplicat obiectelor din ipoteza I pentru a obține rezultatul R ?" este una din problemele delicate ce are un caracter creator, la a cărei rezolvare vor trebui luate în considerație următoarele abordări:

a) rezolvare pe baza unor reguli euristice de tipul

IF (ipoteza este de tipul (I))

and (rezultatul este de tip (R))

THEN se aplică (metoda 1) sau (metoda 2)... sau (metoda n)

b) rezolvare pe baza raționamentelor prin analogie.

c) rezolvare pe baza metodelor de sinteză procedurală.

Problema interogativă de tip *XPR* corespunde formulării "ce obiecte X trebuie prelucrate pentru a obține rezultatul R ?". Problema poate fi rezolvată direct doar dacă P este o bijecție adică știind că $R = P(X)$ soluția va fi $X = P^{-1}(R)$. Dacă însă P nu este o bijecție atunci abordarea trebuie făcută prin metode reductive.

Problemele de tip predicativ au structura similară, componentele lor fiind:

► ipoteza, (I);

► procesul inferențial, (P);

► concluzia, (C).

O astfel de problemă este notată *IPC*. Chiar dacă structural problemele de tip predicativ și cele de tip interogativ sînt asemănătoare, diferă prin faptul că la problemele de tip predicativ ipoteza și concluzia sînt entități cu valoare de adevăr. O problemă predicativă în care concluzia este o necunoscută se numește *derivare*. Derivarea apare ca o problemă rar întîlnită în sisteme inteligente dar se întîlnește frecvent în aplicarea procedurilor ce vizează descompunerea problemelor în subprobleme.

Problema de tip predicativ la care procesul inferențial este necunoscut se numește *demonstrație*. Soluția unei astfel de probleme este formată dintr-un lanț de inferențe ce pornește de la ipoteze și ajunge la concluzie. Aplicarea unui lanț inferențial este caracteristică sistemelor rezolutive pentru demonstrarea automată a teoremelor.

Problema de tip predicativ în care ipoteza este cunoscută se numește "*inducție*". Soluția presupune descoperirea unor concepte inițiale, a cauzalității sau a altor premise prin interpretarea rezultatelor ce se obțin pe baza unor

reguli de inferență cunoscute. Procesul rezolutiv ce implementează metode inductive se mai numește și proces de *învățare* din exemple.

Procesele de tip *imperativ* sînt cele care nu conțin necunoscute în triada ipoteza, proces, rezultat. O problemă de tip imperativ nu necesită date de intrare și rezultatul procesului este aprioric cunoscut. Rezultatul invocării procesului este efectul actual, adică producerea rezultatului cunoscut. Faptul că o problemă este stratificată pe mai multe nivele conceptuale, face ca o problemă la un anumit nivel să se descompună în probleme de alte tipuri pe nivelele inferioare.

Problemele *incomplet formulate* au soluție doar dacă în urma aplicării unor procedee ce nu alterează specificul problemei din enunț, se obține o nouă formulare ce satisface condiția de bună formulare. Principalele procedee ce se aplică în aceste cazuri sînt:

- ♦ reducerea enunțului la o formă de problemă bine formulată ce este făcută prin eliminarea unor elemente lipsite de relevanță, elemente se alterau buna formulare;
- ♦ completarea enunțului de date transmise prin moștenire, aferente pieselor din enunț, cu date asumate prin analogie, cu noile date obținute în urma reformulării de către utilizatorul ce interacționează cu sistemul rezolutiv;
- ♦ descompunerea problemei în subprobleme astfel încît anumite componente să fie bine formulate. Metoda se aplică în general la problemele de complexitate mai mare. Se va obține astfel o rezolvare parțială în situația în care subproblemele complet formulate sînt rezolvate prin procedurile deja descrise;
- ♦ extinderea obiectivelor problemei asupra clasei ce cuprinde problema inițială, introducerea de obiecte abstracte, a condițiilor suplimentare ce permit reformularea enunțului pentru a se satisface condițiile de problemă bine formulată.

Este de la sine înțeles faptul că nu se pune problema rezolvării problemelor incomplet formulate. Pot în schimb exista în anumite cazuri analizoare de problema care să ilustreze inconsistențele de formulare și să justifice refuzul de a rezolva o anumită problemă de către sistem.

5.4. Procesul rezolvării și cunoașterea

Cunoașterea pentru rezolvarea problemei provine din mecanismele inferențiale ale sistemului, din enunțul problemei și din cunoașterea apriori memorată în baza de cunoștințe. Clasificarea cunoașterii despre problemă este o preocupare recentă în inteligența artificială avînd ca scop introducerea unei ordini conceptuale în raportul cunoaștere/rezolvare. Se poate organiza cunoașterea despre problemă folosind concepte ale demonstrării automate a

teoremelor, utilizând conexiunile între matricile formelor prenex. Bibel propune o ierarhizare pe cinci nivele ce au ca bază: cunoașterea factuală, cunoașterea deductivă, cunoașterea rațională, cunoașterea despre structurile globale, cunoașterea despre controlul asupra raționamentelor. Concepută inițial pentru metoda conexiunilor între matrici, clasificarea poate fi extinsă la cazul general fără a ține cont în vreun fel de metoda de rezolvare.

Pornind de la ierarhizarea cunoașterii se obține un arbore de clasificare astfel:

♦ cunoaștere factuală

- conceptuală
- instanțială

♦ cunoaștere procedurală

- transformațională
- inferențială

♦ cunoaștere de control

- pentru structuri
- pentru raționamente

Dependențele reciproce dintre diferitele clase ce formează piesele de cunoaștere pornesc de la cunoașterea factuală ce definește prototipuri ale unor fapte și instante ce descriu fapte individuale. Asupra pieselor de cunoaștere factuală operează cunoașterea procedurală ce specifică proceduri de transformare a pieselor de cunoaștere factuală. Cunoașterea de control asupra procesului de rezolvare se bazează pe piese de cunoaștere procedurală și specifică modul de succesiune în care se înlănțuie procedurile de transformare pentru obținerea de noi structuri, modul de succesiune în aplicarea regulilor de inferență pentru raționamente. Se poate considera că la fiecare nivel pot exista:

♦ piese ale cunoașterii directe;

♦ piese ale cunoașterii deduse în urma aplicării regulilor de inferență, ce sînt diferite de piesele cunoașterii directe. Dacă în baza de cunoștințe există o parte din aceste piese, o serie de procese inferențiale, mari consumatoare de timp, nu mai sînt necesare. Luarea deciziei de introducere în baza de cunoștințe a pieselor de cunoaștere deduse se face după o atentă evaluare a volumului ocupat de acestea în baza de cunoștințe.

În particular, în sisteme expert apare necesitatea de organizare a cunoașterii astfel încît să se țină seama de opiniile mai multor agenți cunoscători, făcînd în același timp posibilă separarea cunoașterii globale comună subiecților cunoscători, de cunoașterea personalizată.

Se tratează mai departe moduri de prezentare a problemelor după diferite formalisme specifice modurilor de reprezentare a cunoștințelor.

5.5. Reprezentarea problemelor în limbajul calculului cu predicate de ordinul întâi

Abordarea problemelor de către sistemul rezolutiv devine posibilă atunci când acestea sînt formulate în acel limbaj de reprezentare ce este înțeles de sistemul ce le va reprezenta. Este deci ușor de înțeles că rezolvarea problemelor exprimate în limbajul calculului de ordinul întâi este adecvată atunci când piesele de cunoaștere sînt reprezentate în acest limbaj. Enunțul problemei este în general exprimat într-un limbaj apropiat de cel natural, ce este transcris în formule ale calculului cu predicate de ordinul întâi.

5.5.1. Particularități ale reprezentării problemelor în limbajul calculului cu predicate de ordinul întâi

Transcrierea și formularea presupune o analiză ce are obiectivele:

- ♦ descompunerea enunțului problemei în piese de cunoaștere ce pot fi descrise cu ajutorul predicatelor de ordinul întâi;
- ♦ extragerea caracteristicilor structurale ale teoriei în care se plasează problema avînd ca scop recunoașterea tipului de similaritate identificatoare a competenței sistemului rezolutiv;
- ♦ definirea cunoașterii factuale prin asignarea de simboluri constante la variabilele predicatelor;
- ♦ definirea obiectivului problemei cu ajutorul formulelor calculului cu predicate de ordinul întâi.

Caracteristicile structurale ale teoriei în care se plasează problema sînt funcție de modalitățile de specificare a componentelor problemei în contextul grupării pieselor de cunoaștere specificate în enunț, de mulțimea suport a obiectelor structurii, de mulțimea simbolurilor funcționale, relaționale și a simbolurilor distincte. Mulțimea suport a structurii cuprinde atît obiectele de tip constantă, organizate în colecții definite fie enumerativ, fie generativ pe baza descrierii unui prototip obiectual și al unui mecanism de generare. Simbolurile funcționale au în general rolul de a furniza termenii ce candidează a fi atașați ca variabile formale sau ca instante, de a furniza termenii posibili ai unor simboluri relaționale în formule. Simbolurile relaționale apar în formulele ce descriu enunțul problemei în limbajul calculului cu predicate de ordinul întâi. Aritatea și ordinul se determină după stabilirea relevanței semantice.

Eficiența sistemului rezolutiv poate fi mărită prin eventuala rejecție a problemelor pentru rezolvarea cărora sistemul nu are competență, înainte de declanșarea raționamentului pentru care tot nu s-ar fi obținut un succes. În

urma transcrierii problemei în formulele calculului cu predicate de ordinul întâi problema se prezintă sistemului în următoarea grupare de simboluri:

- ♦ *suportul obiectual* ce atașează piese de cunoaștere ca variabile formale pentru expresii predicative primare;
- ♦ *simboluri funcționale* formate din piese de cunoaștere pentru definirea termenilor din formule;
- ♦ *simboluri predicative* ce definesc piese cu valoare de adevăr ce descriu proprietăți ale obiectelor sau relații între obiecte;
- ♦ *fapte* - expresii formale ce exprimă adevăruri despre obiectele implicate, cuprinzând reguli specifice teoriei de reguli ale problemei. După unii autori totalitatea formulelor formează *axiomele problemei*;
- ♦ *obiectivul* reprezentat printr-o expresie formală, prin care se exprimă ce se urmărește ca rezultat.

5.5.2. Exemple de reprezentare a problemelor utilizând calculul cu predicate de ordinul întâi

Exemplu 1: Se consideră problema traversării râului de către un grup de excursioniști cu formularea: un grup de excursioniști ajung pe malul stîng al unui râu ce nu poate fi traversat înot. La același mal se găsesc doi copii cu o barcă. În barcă poate să fie la un anumit moment un singur excursionist sau doi copii. Copiii sînt suficient de puternici ca să vîslească de mai multe ori în scopul traversării râului. Starea finală în care se dorește să se ajungă prevede ca grupul excursioniștilor să se găsească pe malul drept, iar cei doi copii împreună cu barca pe malul stîng.

Se propune următoarea grupare de simboluri:

- ♦ *suportul obiectual* format din mulțimea copiilor $C = \{c_1, c_2\}$, mulțimea excursioniștilor $E = \{e_1, e_2, \dots, e_n\}$, elementul barcă $\{b\}$, cele două maluri $\{d, s\}$;
- ♦ *simboluri predicative* ce asertează următoarele proprietăți:
 $C(x)$ cu semnificația că x este un copil, $x \in C$
 $E(x)$ cu semnificația că x este un excursionist, $x \in E$
 $D(x)$ cu semnificația că x este malul drept
 $S(x)$ cu semnificația că x este malul stîng
 $B(x)$ cu semnificația că barca este una din pozițiile $x \in \{s, d\}$
 $T(x, y, z)$ cu semnificația că x traversează cu barca din poziția y în poziția z în care, $x \in CUE$, $y, z \in \{s, d\}$;

- ♦ *faptele* sînt organizate în fapte relevînd condiții ale problemei și fapte relevînd proprietăți ale obiectelor.

Se dau mai jos condițiile și expresiile formale ce definesc problema traversării:

- ▶ barca ajunsă pe malul drept trebuie readusă pe malul stîng numai de către un copil, un excursionist ajuns pe malul drept va rămîne acolo știind că numai așa obiectivul problemei poate fi atins. Condiția de acest tip se scrie:

$$[D(x) \wedge C(x)] \wedge B(d) \rightarrow T(x, d, s)$$

- ▶ dacă barca se găsește pe malul stîng și pe malul drept există un copil care să o readucă înapoi atunci un excursionist poate traversa râul

$$[S(x) \wedge E(x)] \wedge [D(y) \wedge C(y)] \wedge B(s) \rightarrow T(x, s, d)$$

- ▶ dacă cei doi copii se găsesc pe malul stîng aceștia pot traversa râul împreună utilizînd barca, fapt ce se exprimă prin formula:

$$[S(x) \wedge C(x)] \wedge [S(y) \wedge C(y)] \wedge B(s) \rightarrow T(x, s, d) \wedge T(y, s, d)$$

- ▶ traversarea râului de către copii sau excursioniști atrage după sine schimbarea poziției bărcii

$$T(x, y, z) \rightarrow B(z)$$

- ▶ în urma unei traversări obiectele traversate vor fi plasate pe malul destinație

$$T(x, s, d) \rightarrow D(x)$$

$$T(x, d, s) \rightarrow S(x)$$

- ▶ dacă ambii copii sînt pe malul drept unul dintre ei va trebui să aducă barca pe malul stîng

$$[D(x) \wedge C(x)] \wedge [D(y) \wedge C(y)] \wedge B(d) \rightarrow T(x, d, s)$$

Faptele ce exprimă proprietăți ale obiectelor sînt date de mulțimea copiilor, mulțimea excursioniștilor și de poziția inițială a acestora, adică:

$$C(c_1), C(c_2), E(e_1), E(e_2), \dots, E(e_n), S(c_1), S(c_2), S(e_1), \dots, S(e_n), B(s), B(d).$$

- ♦ obiectivele problemei sînt date de poziția finală a copiilor, bărcii și excursioniștilor:

$$D(e_1) \wedge D(e_2) \wedge \dots \wedge D(e_n); S(c_1) \wedge S(c_2) \wedge B(s)$$

Forma generală a formulelor în calculul cu predicate de ordinul întâi nu este convenabilă pentru aplicarea formulelor cunoscute în logică. În acest sens o serie de transformări sînt aplicate acestor formule pentru a fi aduse la forma convenabilă mecanismului de interpretare.

Exemplul 2: Se consideră în continuare problema așezării cuburilor, pentru care modul de reprezentare a cunoașterii a fost tratat în capitolul 4. Se va vedea în paragrafele următoare că pentru multe probleme ce se reprezintă în spațiul stărilor, concepte de căutare în grafuri se vor utiliza pentru descrierea

rezolvării. Green apropie spațiul stărilor de formulele calculului cu predicate introducînd noțiunea de variabilă de stare. Ideea de bază pornește de la descrierea ca stare a situației inițiale, a scopului ca o stare parțială și a unor acțiuni ca operatori ce determină trecerea dintr-o stare în alta. Fiecare stare este reprezentată ca un set finit de formule în limbajul calculului cu predicate de ordinul întâi. Pentru simplitate se va admite că fiecare formulă este atomică. Prezența unei formule într-o stare arată că formula are valoarea adevărat. Fiecare acțiune este reprezentată printr-o condiție de aplicabilitate sau precondiție, adică, o acțiune nu este permisă a se executa în orice stare. Raționamentul se specifică prin două liste, *add_list* ce reține acele formule ce au valoarea adevărat și *delete_list* pentru formulele ce nu mai sînt adevărate în noua stare. Lumea blocurilor este compusă dintr-o masă infinită în sensul că pot fi dispuse oricîte cuburi pe ea. Fiecare bloc are un nume ce este reprezentat în exemplu ca litere ale alfabetului. Starea din fig. 5.6a este descrisă prin următoarele formule:

$$PE_MASA(A) \wedge PE(B, A) \wedge PE(C, B) \wedge PE(D, C) \wedge CLEAR(C)$$

Starea finală din fig. 5.6b va fi descrisă:

$$PE_MASA(D) \wedge PE(C, D) \wedge PE(B, C) \wedge PE(A, B) \wedge CLEAR(A)$$

▼ Fig. 5.6 a) Starea inițială a cuburilor

b) Starea finală a cuburilor

MASA	D	A
	C	B
	B	C
	A	D
a)		b)

5.6. Skolemizarea formulilor

Se vor considera în continuare trei acțiuni posibile, adică trei operatori:
 UNSTACK - așezarea cubului din vârful stivei pe masă;
 STACK - așezarea unui cub aflat pe masă în vârful stivei;
 MOVE - mutarea unui cub dintr-o stivă în alta.
 Acțiunile de acest tip se pot descrie prin

```

Unstack(x)
  Precondiții: PE(x, y), CLEAR(x)
  Add_list:   PE_MASA(x)
  Delete_list: PE(x, y)
Stack(x, y)

```



```

Precondiții: PE_MASA(x), CLEAR(x)
Add_list:    PE(x, y)
Delete_list: PE_MASA(x), CLEAR(y)
Move(x, y)
Precondiții: PE(x, z), CLEAR(x), CLEAR(y)
Add_list:    PE(x, y), CLEAR(z)
Delete_list: PE(x, z), CLEAR(y)

```

Se poate ușor arăta că cea mai bună soluție este dată de succesiunea operatorilor (Unstack(D), Move(C, D), Move(B, C), Stack(A)).

O altă idee este de a considera un plan neliniar cu scopuri nerealizate și acțiuni parțial descrise ca reprezentări implicite a setului de planuri complete (complet ordonate, cu acțiuni descrise complet și scopuri neatinse). Se va spune că propoziția p este necesar adevărată (scris $\Box p$) în pasul s dacă și numai dacă p este adevărată în s la fiecare completare a planului. Similar, o propoziție p este posibil adevărată (scris $\Diamond p$) în pasul s dacă și numai dacă p este adevărat în s la cel puțin o completare a planului. Aceleași definiții sînt aplicabile în succesiunea pașilor spunînd că pasul s_1 este necesar înainte de s_2 (scris $\Box (s_1 < s_2)$), dacă este cazul la fiecare completare și similar pentru posibilitate.

Cu restricțiile descrise, criteriul pentru necesar adevărat se reprezintă simplu. Astfel p este necesar adevărat în pasul s dacă și numai dacă:

1. există un pas t după s în care p este necesar adevărat;
2. pentru toți pașii C posibili, înainte de s și negînd p există un pas W necesar între C și s , asertînd p oricînd C neagă p .

Pasul t este numit *stabilizator*, C este numit *distrugător* și W este numit *reparator*. De aceea definiția arată că o propoziție este necesar adevărată dacă este stabilită oricînd, și pentru orice distrugător posibil există un refăcător care o face adevărată din nou. De aceea complexitatea algoritmului este $O(n^3)$.

Realizarea scopului înseamnă să facă aceasta necesar adevărată în pasul dat. De aceea, criteriul de adevăr poate fi astfel utilizat pentru procesul de planificare dacă există capacitatea interpretării sale procedurale. Pentru aceasta se va scrie ca o formulă logică așa cum se vede în fig. 5.7.

▼ Fig. 5.7. Formula logică

$$\begin{array}{lcl}
 \exists t \ t \leq s \wedge \Box \text{assert}(p, t) & \wedge & \\
 \forall c \ \Box s \leq c & \vee & \\
 \forall q \ \Box \text{negate}(c, p) & \vee & \\
 \exists w \ \Box c < w \leq s & \wedge & \\
 \text{assert}(p, w) & &
 \end{array}$$

Conectivile au următoarea interpretare:

- \wedge : ambele drumuri trebuie să fie adevărate;
- \vee : alegere nedeterministă între cele două căi;
- \exists : alegerea unui element, fie unul existent fie unul nou;
- \forall : o buclă cu toate posibilitățile.

- ♦ rescrierea formulelor utilizând forma Skolem conjunctivă, mult mai convenabilă pentru demonstrarea automată.

Se spune că o formulă f are o formă Skolem f^s ce nu este logic echivalentă, dar este adevărată dacă și numai dacă f este adevărată.

Reducerea numărului de conective are ca obiect aducerea formulelor la forma convenabilă aplicării regulilor de inferență. În teoria inferențelor logice reprezentarea regulilor se face în formatul denumit și forma clauzală de tipul

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \vdash B_1 \vee B_2 \vee \dots \vee B_m$$

În situația unui test de nesatisfiabilitate se obține clauza vidă de forma

$$(A_1 \wedge A_2 \wedge \dots \wedge A_n) \vdash \square$$

Considerînd că A_1, A_2, \dots, A_m sînt formule ce se presupun a fi independente, pentru fiecare există o clauză A_i , $1 \leq i \leq n$. Se dau mai jos cîteva reguli de calcul în formule ale limbajului de calcul cu predicate de ordinul întîi

$$(A \rightarrow B) \vdash (\neg A \vee B)$$

$$\neg(A \rightarrow B) \vdash (A \wedge \neg B)$$

$$(A = B) \vdash (A \vee \neg B) \wedge (\neg A \vee B).$$

Fie x o variabilă și $A, B, A(x)$ și $B(x)$ formule, în care A și B nu conțin apariții libere ale variabilei x , atunci:

$$\vdash \forall x (A \rightarrow B(x)) \equiv A \rightarrow \forall x B(x)$$

$$\vdash \forall x (A(x) \rightarrow B) \equiv \exists x A(x) \rightarrow B$$

$$\vdash \exists x (A \rightarrow B(x)) \equiv A \rightarrow \exists x B(x)$$

$$\vdash \exists x (A(x) \rightarrow B) \equiv \forall x A(x) \rightarrow B$$

Pornind de la o formulă oarecare A se poate găsi o formulă A' numită și formă prenex a lui A cu proprietățile:

- ♦ A' este echivalentă cu A adică $\vdash A \equiv A'$;

- ♦ în formula A' , cuantificatorii sînt plasați în partea cea mai din stînga a formulei prefixînd corpul formulei în care se găsesc simbolurile conectivelor logice ce figurează în scopul oricărui cuantificator.

Reguli de transformare. Dacă x și y sînt variabile distincte, $A, B, A(x), B(x), A(x, y)$ formule, iar A și B nu conțin apariții libere ale lui x și dacă x este liber pentru y în $A(x, y)$ atunci:

$$\vdash \forall x A \equiv A$$

$$\vdash \exists x A \equiv A$$

$$\vdash \forall x \forall y A(x, y) \equiv \forall y \forall x A(x, y)$$

$$\vdash \exists x \exists y A(x, y) \equiv \exists y \exists x A(x, y)$$

$$\forall x \forall y A(x, y) \vdash \forall x A(x, x)$$

$$\exists x A(x, x) \vdash \exists x \exists y A(x, y)$$

$$\forall x A(x) \vdash \exists x A(x)$$

$$\exists x \forall y A(x, y) \vdash \forall y \exists x A(x, y)$$

$$\vdash \exists x A(x) \equiv \neg \forall x \neg A(x)$$

$$\vdash \forall x A(x) \equiv \neg \exists x \neg A(x)$$

$$\vdash \neg \forall x A(x) \equiv \exists x \neg A(x)$$

$$\begin{aligned}
&\vdash \neg \exists x A(x) \equiv \forall x \neg A(x) \\
&\vdash \forall x A(x) \wedge \forall x B(x) \equiv \forall x (A(x) \wedge B(x)) \\
&\vdash \exists x A(x) \vee \exists x B(x) \equiv \exists x (A(x) \vee B(x)) \\
&\vdash A \wedge \forall x B(x) \equiv \forall x (A \wedge B(x)) \\
&\vdash A \vee \exists x B(x) \equiv \exists x (A \vee B(x)) \\
&\vdash A \wedge \exists x B(x) \equiv \exists x (A \wedge B(x)) \\
&\vdash A \vee \forall x B(x) \equiv \forall x (A \vee B(x)) \\
&\exists x (A(x) \wedge B(x)) \vdash \exists x A(x) \wedge \exists x B(x) \\
&\forall x A(x) \vee \forall x B(x) \vdash \forall x (A(x) \vee B(x)).
\end{aligned}$$

Obținerea formei prenex are loc după următoarea succesiune de operații:

- ♦ tranzitarea negației de la formule la atomi. Dacă x este o variabilă, iar $A, B, A(x)$ și $B(x)$ sînt formule atunci:

$$\begin{aligned}
&\neg \forall x A(x) \equiv \exists x \neg A(x) \\
&\neg \exists x A(x) \equiv \forall x \neg A(x) \\
&\neg (A \wedge B) \equiv \neg A \vee \neg B \\
&\neg (A \vee B) \equiv \neg A \wedge \neg B \\
&\neg \neg A \equiv A;
\end{aligned}$$

- ♦ transferul cuantificatorilor de la formule la atomi. Dacă x este o variabilă $A(x)$ și B sînt formule cu x liberă în A dar legată în B atunci:

$$\begin{aligned}
&\forall x (A(x) \vee B) \vdash \forall x A(x) \vee B \\
&\forall x (B \vee A(x)) \vdash B \vee \forall x A(x) \\
&\forall x (A(x) \wedge B) \vdash \forall x A(x) \wedge B \\
&\forall x (B \wedge A(x)) \vdash B \wedge \forall x A(x) \\
&\exists x (A(x) \vee B) \vdash \exists x A(x) \vee B \\
&\exists x (B \vee A(x)) \vdash B \vee \exists x A(x) \\
&\exists x (B \wedge A(x)) \vdash B \wedge \exists x A(x) \\
&\exists x (A(x) \wedge B) \vdash \exists x A(x) \wedge B.
\end{aligned}$$

Dacă x este o variabilă, iar $A(x)$ și $B(x)$ sînt formule cu x liberă atît în $A(x)$ cît și în $B(x)$ atunci:

$$\begin{aligned}
&\forall x (A(x) \wedge B(x)) \vdash \forall x A(x) \wedge \forall x B(x) \\
&\exists x (A(x) \vee B(x)) \vdash \exists x A(x) \vee \exists x B(x).
\end{aligned}$$

Dacă x și y sînt variabile $A(x, y)$ și $B(x, y)$ sînt formule, x este legată în cel puțin una din formulele $A(x, y)$ și $B(x, y)$, iar y este liberă atît în $A(x, y)$ cît și în $B(x, y)$, atunci:

$$\begin{aligned}
&\forall x \forall y (A(x, y) \vee B(x, y)) \vdash \forall y \forall x (A(x, y) \vee B(x, y)) \\
&\exists x \exists y (A(x, y) \wedge B(x, y)) \vdash \exists y \exists x (A(x, y) \wedge B(x, y));
\end{aligned}$$

- ♦ schimbarea numelor variabilelor. Cînd doi cuantificatori prefixează același nume de variabilă, numele variabilei este înlocuit cu un simbol diferit de celelalte simboluri pentru variabilele utilizate. Procedura se aplică în mod repetat pînă se obține individualizarea simbolurilor pentru variabile în formula considerată. Procedul se mai numește și standardizarea variabilelor;

- ♦ eliminarea cuantificatorului existențial. Acest proces se bazează pe înlocuirea variabilelor cuantificate de acesta prin funcții Skolem.

Fie f o formulă în limbajul calculului cu predicate de ordinul întâi avînd variabilele libere x_1, x_2, \dots, x_n, y . Pentru o variabilă oarecare, (de exemplu y) se poate asocia un simbol funcțional $f_f(x_1, x_2, \dots, x_n)$ denumit și funcție Skolem a lui f ce satisface axioma Skolem a lui f .

$$\models x_1, x_2, \dots, x_n (\exists y f(x_1, x_2, \dots, x_n, y) \rightarrow f(x_1, x_2, \dots, x_n, f_f(x_1, x_2, \dots, x_n))).$$

Astfel se elimină din formula dată cuantificatorul existențial prin înlocuirea variabilei cuantificată cu un simbol funcțional inexistent avînd ca argumente acele variabile ce sînt cuantificate universal. Se remarcă faptul că simbolurile de variabile cuantificate existențial, ce nu figerează în scopul nici unui cuantificator universal, reprezintă funcții 0-are. Astfel formula $\exists v \forall x \forall y \exists z P(x, y, z, v)$ este echivalentă cu $\forall x \forall y P(x, x, f(x, y), a)$, în care, variabila z cuantificată existențial a fost înlocuită cu funcția Skolem $f(x, y)$ cu argumentele cuantificate universal în al căror scop se află cuantificarea existențială $\exists z$, variabila v cuantificată existențial a fost înlocuită cu o funcție Skolem $f() = a$ (constantă), întrucît cuantificarea existențială $\exists v$ nu se află în scopul nici unei cuantificări universale, așa că funcția Skolem este de aritate zero;

- ♦ tranzitarea în prefix a cuantificatorilor universali. Întrucît simbolurile de variabile sînt individualizate pot fi grupați în prefix toți cuantificatorii universali obținînd astfel forma normală prenex a formulei. Un cuantificator ce are ca scop o subformulă are același scop și dacă prefixează întreaga formulă.

În concluzie, o formulă în forma normală prenex este alcătuită din două părți, și anume:

- ♦ partea din stînga ce a fost denumită și prefix, și cuprinde toți cuantificatorii universali ai formulei;
- ♦ partea plasată după prefix numită și matricea formulei ce cuprinde literale libere de cuantificatori și alcătuiește corpul propriu-zis al formulei.

Forma normală prenex permite transcrierea subformulelor componente în vederea obținerii formei normale conjunctive, numită și forma Skolem conjunctivă, formată din conjuncții de disjuncții. Dacă A, B, C sînt formule, atunci:

$$\models A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

$$\models (A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

Astfel, forma Skolem conjunctivă trebuie eventual simplificată pentru a se ajunge la cea mai condensată formă de exprimare a formulei. Procedura ce duce la forma condensată implică următorii pași:

- ♦ se elimină tautologiile;
- ♦ dacă se întîlnește subformule de tipul $\alpha \vee A \vee \alpha$, cu α un literal și A o formulă este echivalentă cu $\alpha \vee A$;
- ♦ dacă A, B și C sînt clauze cu $\forall B$ și $\forall C$ închiderea universală a clauzelor B și C adică $\models \forall B \rightarrow \forall C$ se pot aplica regulile de

transcriere $B \wedge A \wedge C \vdash B \wedge A$ și $C \wedge A \wedge B \vdash A \wedge B$, și despre clauza C se spune că este subsumată clauzei B .

Transcrierea în notație de mulțime clauzală se face pentru o utilizare mai ușoară a formulelor Skolem aplicând criterii de retranscriere. În acest scop cuantificatorii universalii ce prefixează formula sînt șterși și sînt considerați implicit pentru toate simbolurile de variabilă, variabilele sînt legate și prin standarizarea simbolurilor ordinea în care sînt înscriși nu mai este relevantă la interpretare. Simbolurile \vee sînt omise, simbolurile \wedge sînt înlocuite prin virgule, parantezele cu excepția celor pentru variabilele funcțiilor sînt omise. Formula $\forall x \forall y \forall z ((P(x) \vee Q(y)) \wedge R(z))$ se va scrie $Px Qy, Rz$ notație ce se numește și mulțime cauzală.

5.7. Reprezentarea în spațiul stărilor

Reprezentarea problemei în spațiul stărilor presupune luarea în considerație a stărilor și transformărilor acestora datorate unui număr finit de operatori. Matematic se poate considera scrierea ca reprezentată prin tripletul $P = (S, G, R)$ în care,

- ▶ S reprezintă mulțimea stărilor;
- ▶ $G \subset S$ mulțimea obiectivelor problemei;
- ▶ $R \subset S \times S$ mulțimea de transformări de stare ce indică un drum în "graful problemei".

Se spune că secvența de stări (s_0, s_1, \dots, s_n) formează un drum în graf dacă $s_i s_{i+1} \in R \ (\forall) \ i = \overline{0, n}$, iar n -uplul (s_0, s_1, \dots, s_n) reprezintă o soluție a problemei pentru s_0 , dacă $s_n \in G$.

Cunoașterea despre problemă este definită prin:

- ♦ mulțimea obiectelor abstracte ale problemei;
- ♦ mulțimea de operatori;
- ♦ starea inițială;
- ♦ starea finală formată din stări ce reprezintă obiectivul problemei.

Se definește obiectivul problemei, ca o structură simbolică ce reprezintă o stare de lucruri care odată atinsă confirmă faptul că drumul ce a fost parcurs de la starea inițială prin graful problemei este o soluție. Un alt mod de a privi obiectivul este ca o procedură pentru testarea comportării ce oferă criterii de apreciere în căutarea soluțiilor prin intermediul spațiului stărilor. În procesul de rezolvare a problemei, starea problemei reprezintă o informație dinamică cu un conținut care evoluează ca urmare a mecanismelor de rezolvare. Pot fi stabilite cîteva obiective în procesul rezolvării, și anume:

- ♦ parcurgerea înainte în spațiul stărilor cu scopul atingerii obiectivului;

- ♦ parcurgerea înapoi avînd ca scop revenirea la punctele anterioare pentru comutarea procesului de rezolvare pe o altă alternativă de drum.

Informația de stare conține acele elemente ale cunoașterii prin care:

- ♦ se asigură tranziția de la o stare la alta, înainte și înapoi;
- ♦ se asigură informația de acces la memorie pentru oricare din stările curente;
- ♦ se asigură informația de revenire pentru refacerea stării sistemului în punctele de ramificație importante ale grafului și selectarea unei alternative în caz de eșec;
- ♦ se asigură informația de acces la alte trasee ale grafului în situația prelucrărilor paralele;
- ♦ păstrarea informațiilor de justificare a pașilor sistemului de rezolvare.

Prin spațiul stărilor se înțelege o mulțime finită de obiecte implicate în rezolvarea problemei pornind de la starea inițială prin aplicarea unei mulțimi finite de operatori. Aplicarea unui operator asupra unei stări are ca efect producerea unei stări noi sau a unui număr finit de stări alternative. Rezolvarea presupune căutarea în spațiul stărilor a unei stări ce corespunde obiectivului problemei. Parcurgerea grafului de la starea inițială s_0 la starea finală s_n se face prin alegerea unui drum dintr-un număr finit de alternative. În acest proces de parcurgere este cunoscut criteriul de apreciere a obiectivului fără a se cunoaște ordinea de aplicare a operatorilor sau lungimea drumului de parcurs.

Se consideră ca exemplu o zonă închisă de 9 spații în care sînt așezate un număr de 8 cutii și un spațiu gol. Se cere să se găsească modul de mișcare a cutiilor în zona delimitată cu un singur spațiu liber pentru ca o anumită cutie să ajungă dintr-o poziție inițială dată într-o poziție finală dată, celelalte cutii găsindu-se în poziții nestabilite. Presupunînd descrierea stării sub forma matricială în care a fost notată cu e poziția golului starea inițială a problemei este dată în fig. 5.9a iar starea finală în fig. 5.9b.

▼ Fig. 5.9 a) Starea inițială
b) Starea finală

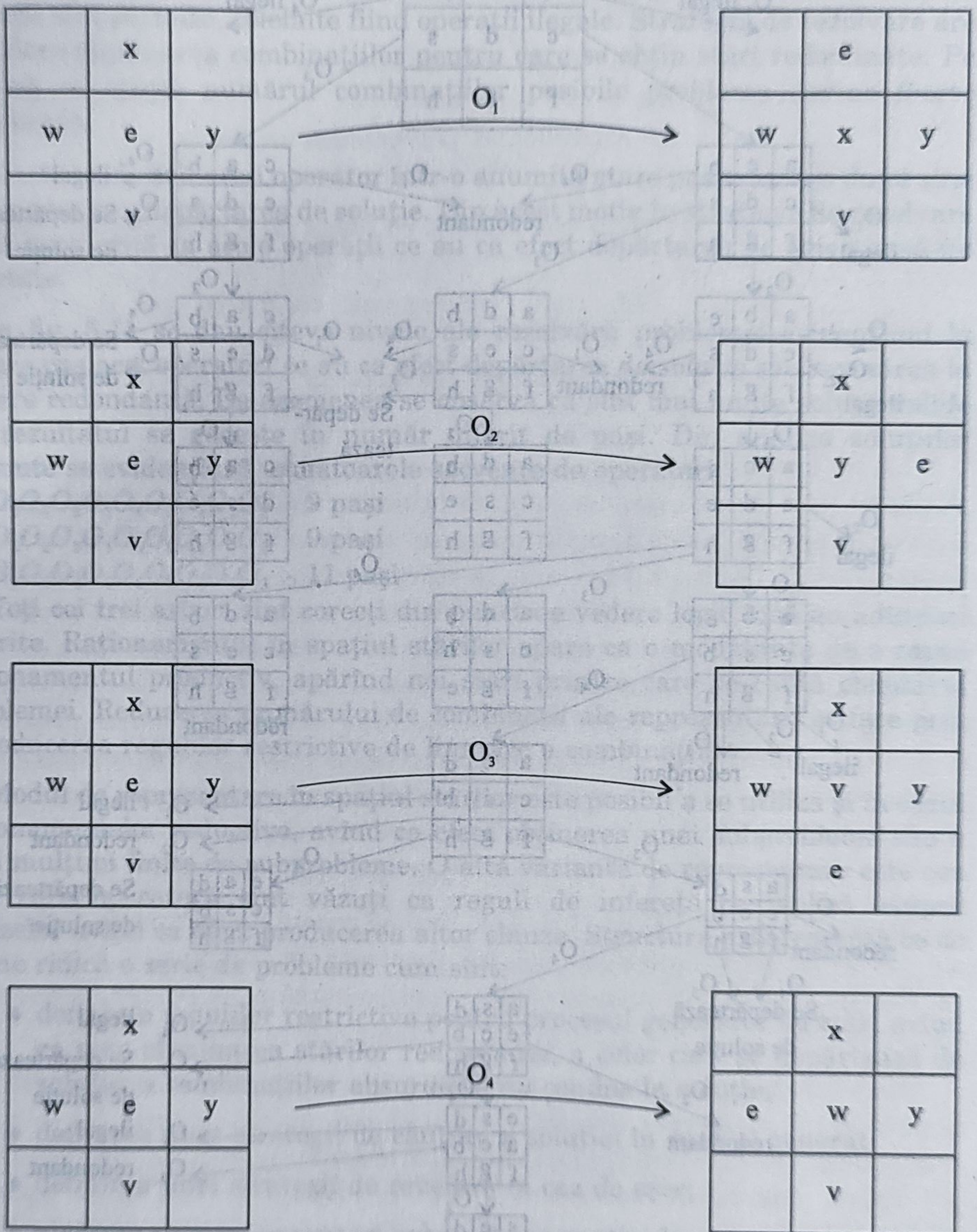
a)

$i \backslash j$	1	2	3
1	e	a	b
2	c	d	s
3	f	g	h

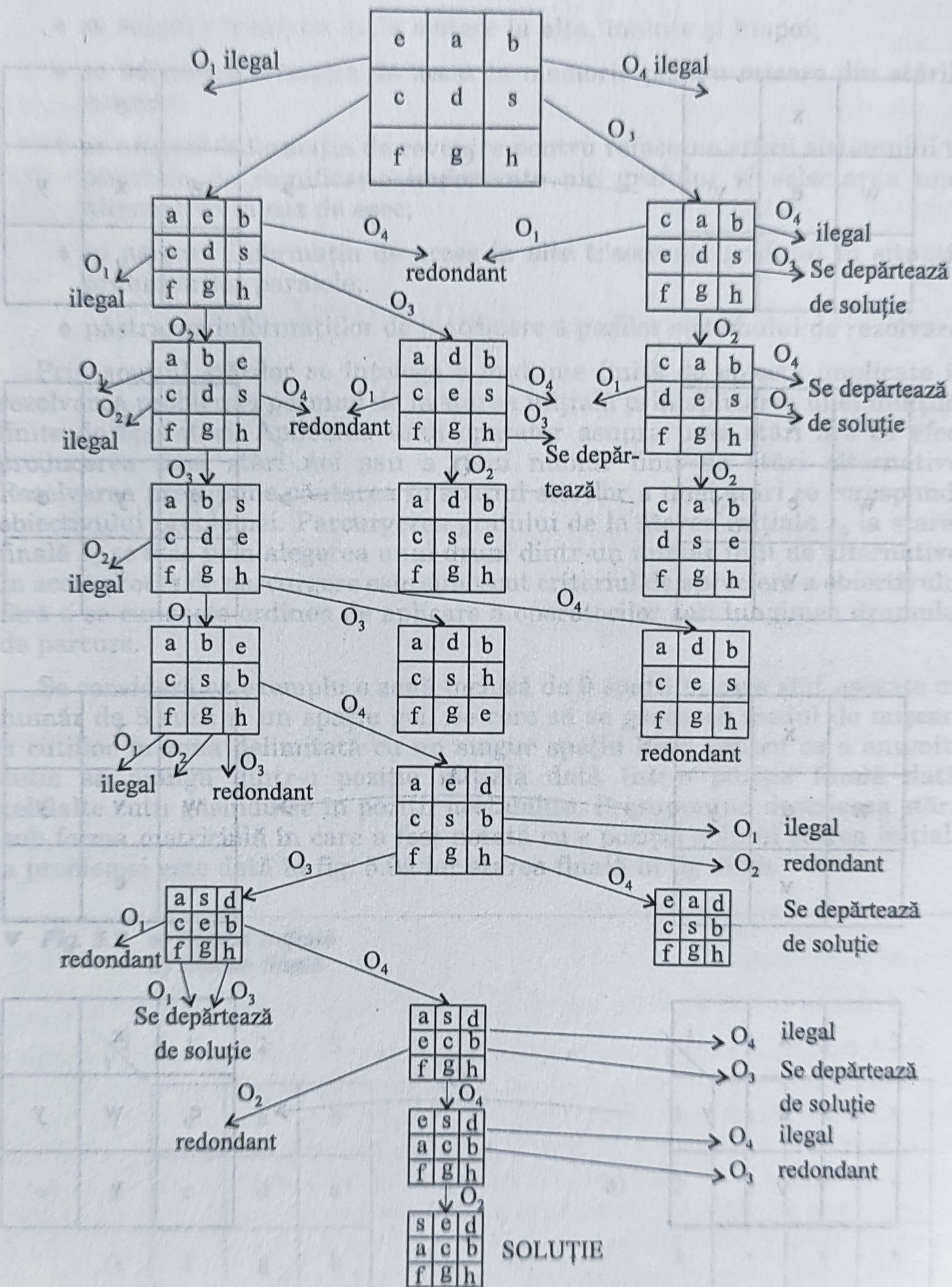
b)

$i \backslash j$	1	2	3
1	s	*	*
2	*	*	*
3	*	*	*

▼ Fig. 5.10. Operatori



▼ Fig. 5.11. Pași de rezolvare a problemei



Obiectivul problemei este reprezentat de starea finală a cutiei s , pozițiile celorlalte cutii și a golului sînt marcate prin $*$.

În strategia de rezolvare se consideră patru operatori notați O_1, O_2, O_3, O_4 pentru deplasarea golului, operatori ce sînt explicitați în spațiul stărilor ca în fig. 5.10. Într-o anumită stare funcție de poziția golului numai anumite operații sînt permise, celelalte fiind operații ilegale. Strategia de rezolvare are în vedere eliminarea combinațiilor pentru care se obțin stări redondante. Pe măsură ce crește numărul combinațiilor posibile problema devine foarte complicată.

Efectul aplicării unui operator într-o anumită stare poate atrage după sine apropierea sau depărtarea de soluție. Din acest motiv în strategia de rezolvare se va avea grijă ca acele operații ce au ca efect depărtarea de obiectiv să fie rejectate.

În fig. 5.11 se dau cîteva nivele ale rezolvării problemei evidențiind la fiecare pas acei operatori ce au ca efect depărtarea de soluție sau revenirea la o stare redondantă. De asemenea se observă că sînt mai multe soluții valide iar rezultatul se găsește în număr diferit de pași. Din analiza soluțiilor obținute se evidențiază următoarele secvențe de operatori:

$O_2 O_3 O_2 O_1 O_4 O_3 O_4 O_1 O_2$

9 pași

$O_3 O_2 O_2 O_1 O_4 O_3 O_4 O_1 O_2$

9 pași

$O_2 O_2 O_3 O_1 O_4 O_3 O_4 O_1 O_2$

11 pași

Toți cei trei arbori sînt corecți din punct de vedere logic însă au adîncimi diferite. Raționamentul în spațiul stărilor apare ca o modalitate de a capta raționamentul productiv, apărînd noi stări printre care se caută obiectivul problemei. Reducerea numărului de combinații ale reprezentării se face prin introducerea regulilor restrictive de limitare a combinațiilor.

Modul de reprezentare în spațiul stărilor este posibil a se utiliza și în cazul raționamentelor reductive, avînd ca efect obținerea unei subprobleme sau a unei mulțimi finite de subprobleme. O altă variantă de reprezentare este cea prin care operatorii sînt văzuți ca reguli de inferență, se aplică asupra clauzelor avînd ca efect producerea altor clauze. Structura arborescentă ce se obține ridică o serie de probleme cum sînt:

- ♦ definirea regulilor restrictive pentru procesul generator de stări avînd ca scop eliminarea stărilor redondante, a celor care se depărtează de soluție, a combinațiilor absurde ce nu conduc la soluție;
- ♦ definirea unor strategii de căutare a soluției în spațiul generat;
- ♦ definirea unei strategii de revenire în caz de eșec;
- ♦ alegerea unei reprezentări compacte ca spațiu de memorie ocupat.

Se prezintă mai jos problema traversării râului pentru a cărei rezolvare se va utiliza spațiul stărilor. Notînd cu S și D stări parțiale ale problemei ce reprezintă situația pe cele două maluri, stîng și respectiv drept, se obține starea ca o reuniune a celor două stări parțiale $S \cup D$.

starea inițială: $S_0 = \{c_1, c_2, e_1, e_2, \dots, e_n, b\}$

$$D_0 = \emptyset$$

starea finală: $S_f = \{c_1, c_2, b\}$

$$D_f = \{e_1, e_2, \dots, e_n\}$$

operatori: Ted - un copil traversează cu barca pe malul drept $\{c_j, b\} \subset s_j$,
 $j = 1, 2$

T2cd - doi copii traversează cu barca pe malul drept
 $\{c_1, c_2, b\} \subset s_i$

Ted - un excursionist traversează cu barca pe malul drept
 $\{e_j, b\} \subset s_i, \overline{j=1, n}$

Tcs - un copil traversează cu barca pe malul stîng $\{c_j, b\} \subset d_j$,
 $j = 1, 2$

T2cs - doi copii traversează cu barca pe malul stîng
 $\{c_1, c_2, b\} \subset d_i$

Tes - un excursionist traversează cu barca pe malul stîng
 $\{e_j, b\} \subset d_i, \overline{j=1, n}$.

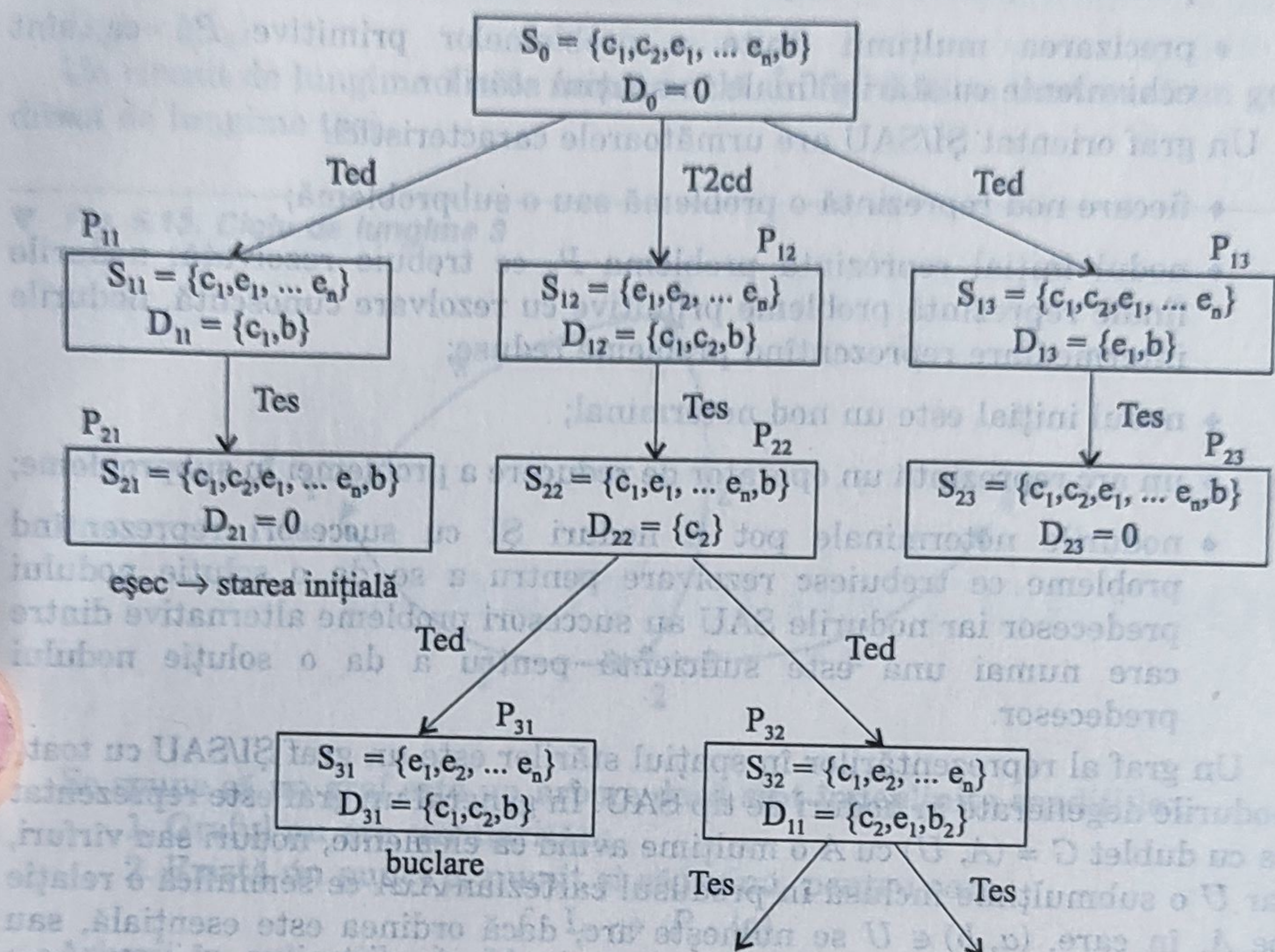
Prin aplicarea operatorilor definiți mai sus se poate genera spațiul stărilor, spațiu în care prima stare este starea inițială. Similar cu problema așezării cutiilor în spațiul închis anumiți operatori sînt invalizi în sensul că nu pot fi aplicați stării curente, alții conduc la stări redondante sau se depărtează de soluție. În fig. 5.12 se arată modul de formare a arborelui asociat problemei de traversare a râului pentru primele trei nivele. Graful de tranziție a stărilor cu punctul de plecare stare inițială are ramuri ce conduc la obiectivul final, operatorii ce conduc la stări redondante fiind eliminați. Soluția problemei poate fi privită fie ca o succesiune de stări fie ca o secvență a operatorilor ce conduc la starea obiectiv. Furnizarea soluției ca o secvență de operatori ce conduce la obiectiv va da

$$O = \{T2cd, Tcs, Ted, Tcs, T2cd, Tcs, Ted, Tcs, T2cd, \dots\}$$

în care secvența $O' = \{T2cd, Tcs, Ted, Tcs\}$ se repetă de un număr de ori egal cu numărul excursioniștilor. Soluția furnizată în acest mod este soluția oricărei probleme de traversare cu doi copii, o barcă și un număr oarecare de excursioniști.

Din analiza reprezentării în spațiul stărilor se observă o asemănare între aceasta și reprezentarea sistemelor de producții. Astfel, un operator se aplică numai dacă anumite criterii prealabile sînt îndeplinite de starea asupra căreia se aplică operatorul (echivalentă cu preconditionia), rezultatul aplicării operatorului fiind obținerea unei stări noi din cea veche (echivalentul acțiunii sau producției). Strategia de control este cea care determină selectarea operatorilor (regulilor de producție) și a modului de aplicare succesivă a acestora pentru generarea spațiului stărilor.

▼ Fig. 5.12. Arbore asociat problemei traversării râului



Aplicarea unui operator asupra unei stări fără a putea reveni la starea anterioară pentru a apela un alt operator formează *strategia de control irevocabilă*. Strategia de control se numește *tentativă* dacă la un anumit punct se poate lua decizia de revenire la o stare precedentă în vederea aplicării de noi operatori. Strategia de control cu *revenire* (backtracking) este acea strategie prin care punctul de reluare este stabilit din momentul selectării operatorului, așa că în caz de eșec se poate relua procesul de la un astfel de punct, nu neapărat cel mai apropiat. Strategia de control prin *căutare de grafuri* este cea prin care se păstrează informații privind consecințele aplicării operatorilor la fiecare pas al generării.

5.8. Utilizarea grafurilor ȘI/SAU

Aplicarea strategiilor de rezolvare reductive asupra unei probleme P_0 pornește de la următoarele precizări:

- ♦ descrierea problemei inițiale P_0 exprimată în spațiul stărilor prin starea inițială P_0 ;

- ♦ precizarea unei mulțimi finite de operatori care aplicați asupra problemei produc o mulțime finită de subprobleme;
- ♦ precizarea mulțimii finite a problemelor primitive P_p , ce sînt echivalente cu stările finale în spațiul stărilor.

Un graf orientat ȘI/SAU are următoarele caracteristici:

- ♦ fiecare nod reprezintă o problemă sau o subproblemă;
- ♦ nodul inițial reprezintă problema P_0 ce trebuie rezolvată, nodurile finale reprezintă probleme primitive cu rezolvare cunoscută, nodurile intermediare reprezentînd probleme reduse;
- ♦ nodul inițial este un nod neterminal;
- ♦ un arc reprezintă un operator de reducere a problemei în subprobleme;
- ♦ nodurile neterminale pot fi noduri ȘI cu succesori reprezentînd probleme ce trebuiesc rezolvare pentru a se da o soluție nodului predecesor iar nodurile SAU au succesori probleme alternative dintre care numai una este suficientă pentru a da o soluție nodului predecesor.

Un graf al reprezentărilor în spațiul stărilor este un graf ȘI/SAU cu toate nodurile degenerate în noduri de tip SAU. În general un graf este reprezentat ca cu dublet $G = (A, U)$ cu A o mulțime avînd ca elemente, noduri sau vîrfuri, iar U o submulțime inclusă în produsul cartezian $A \times A$ ce semnifică o relație pe A , în care, $(a, b) \in U$ se numește arc, dacă ordinea este esențială, sau muchie dacă ordinea nodurilor nu este importantă. Două noduri a_i și a_j sînt adiacente dacă există un arc sau o muchie ce leagă a_i de a_j . O secvență de noduri și arce adiacente ce sînt parcurse în același sens formează un drum. Se poate deci defini drumul prin succesiunea nodurilor $d = (a_1, a_2, \dots, a_n)$ cu proprietatea că $(a_1, a_2), \dots, (a_{n-1}, a_n)$ sînt adiacente. Nodul a_1 este capătul inițial al drumului iar nodul a_n este capătul final al drumului. Dacă $a_1 = a_n$ atunci drumul definește un circuit. Numărul de arce al unui drum definește lungimea drumului. Se numește drum elementar acel drum care trece o singură dată printr-un nod, iar drum simplu un drum ce nu conține de două ori același arc. Pentru un graf neorientat se numește lanț, o succesiune (x_1, x_2, \dots, x_n) , ciclu în lanț dacă $x_1 = x_n$, lanț elementar dacă nodurile sînt distincte, lanț simplu dacă toate muchiile sînt distincte.

Se spune că un graf G este conex dacă oricare dintre două noduri sînt extremități ale unui lanț în G . Un graf conex fără cicluri este un arbore. Într-un graf se numește arborescență de rădăcină a , un graf orientat în care orice nod $a_i \neq a$ este o extremitate pentru un drum unic în care cealaltă extremitate este a .

Fundamentală pentru discuțiile ulterioare este noțiunea de circuit într-un graf. Intuitiv, un circuit este o buclă închisă formată din arce care traversează în secvență punctele circuitului pornind de la un punct de pornire. Riguros, dacă (P, L, f, b) este un graf direct cu funcția de înaintare f și de revenire b , arcele distincte x_1, x_2, \dots, x_n formează un circuit de lungime n dacă:

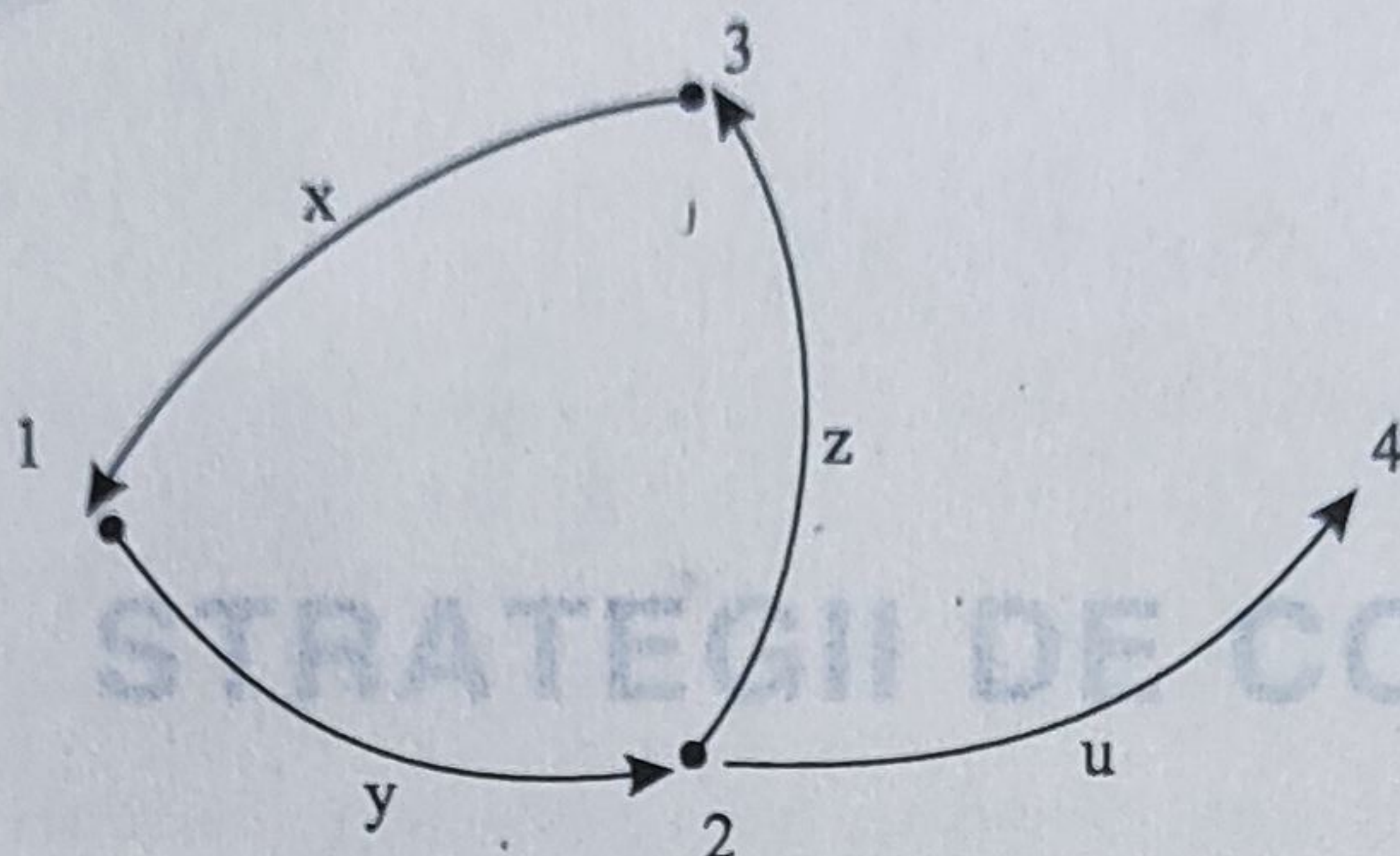
$$f(x_1) = b(x_2)$$

$$f(x_2) = b(x_3)$$

$$f(x_n) = b(x_1)$$

Un circuit de lungime 1 este numit buclă. În fig. 5.13 se ilustrează un graf direct de lungime trei.

▼ Fig. 5.13. Ciclu de lungime 3



Se spune că un graf este un arbore dacă sînt îndeplinite condițiile:

1. Graful nu are circularități;
2. Există un punct p , numit și rădăcină, pentru care

$$f: L \rightarrow P - \{p\}$$

Arborii în aplicațiile de IA sînt reprezentați cu rădăcina sus și arcele orientate în jos. Din acest motiv săgețile se pot omite întrucît totdeauna sînt orientate de sus în jos. Frunzele arborelui sînt acele noduri q din P de la care nu pornesc arce spre alte noduri. Caracterizarea cea mai uzuală a adîncimii unui nod într-un arbore este nivelul nodului. Subsetul P_1 al punctelor $f(x_i)$ din P pentru care

$$b(x_1) = b(x_2) = \dots = b(x_n) = p$$

constituie nivelul 1. Atunci:

$$P_1 = \{f(x_1), f(x_2), \dots, f(x_n)\}.$$

Setul P_k al punctelor nivelului k este definit recursiv funcție de nivelul $k - 1$. Astfel, dacă $b(x_i) \in P_{k-1}$ pentru $i = 1, 2, \dots, j$ atunci:

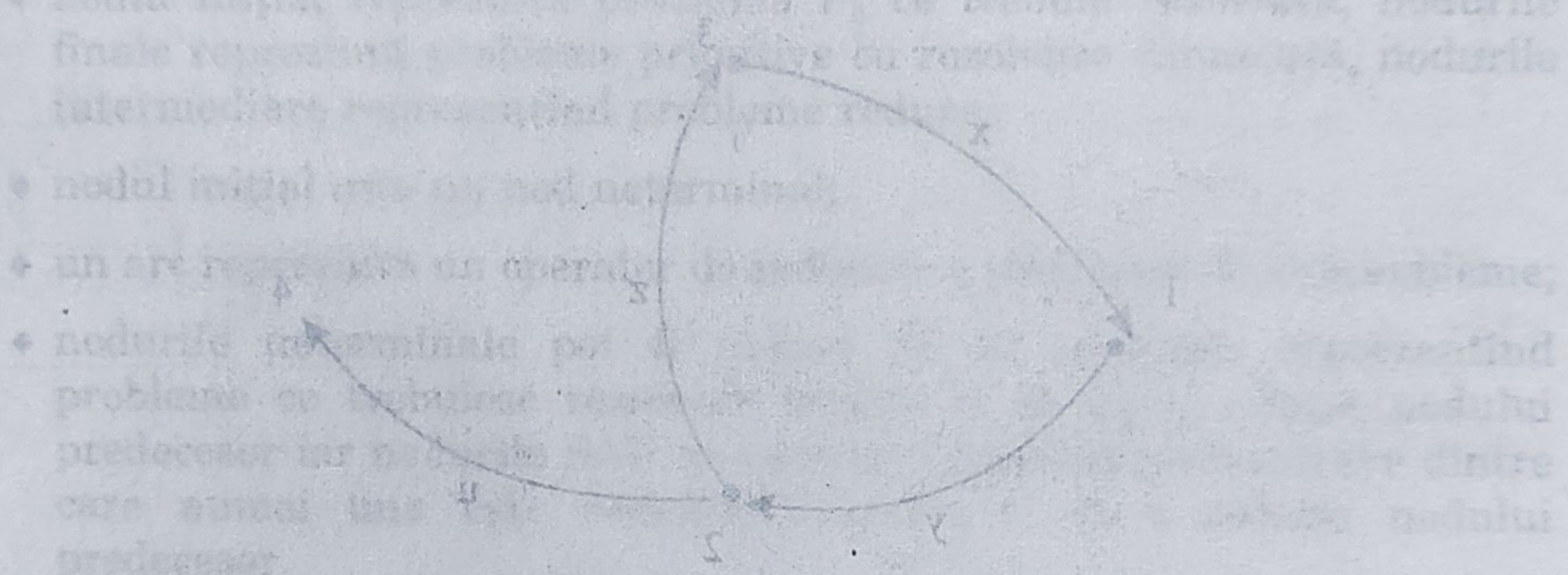
$$P_k = \{f(x_i): i = 1, 2, \dots, j\}.$$

Este important de reținut faptul că mașina privește graful altfel decît omul, în sensul că pentru mașină singurul acces la graf este asociat cvadruplului (P, L, f, b) .

Ca exemplu, pentru calculul integralei definite se poate aplica tehnica descompunerii în serie, integrarea prin părți, schimbarea de variabilă. Fiecare din aceste metode conduce la soluție validă, deci pornind de la starea inițială (problema P_0) un arbore SAU indică necesitatea obținerii unei soluții fără a fi necesară revenirea și reluarea unei alte ramuri a arborelui. Rezolvarea după generarea arborelui ȘI/SAU de reducere a problemei presupune căutarea în graf a condițiilor necesare ca nodul inițial reprezentînd problema să fie

solvabil. Se spune că un nod este solvabil dacă nodul este un nod terminal, deci o problemă primitivă, dacă nodul este un nod neterminal de tip ȘI ce are toate nodurile succesoare solvabile, dacă nodul este nod neterminal de tip SAU și cel puțin un nod succesori este solvabil.

▼ Fig. 2.12. Ciclu de lungime 3



6

STRATEGII DE CONTROL

Am văzut în capitolul precedent că o problemă poate fi rezolvată prin aplicarea unei strategii de control. Această strategie este dependentă de problema în sine și de informațiile disponibile. Cea mai simplă strategie este cea a operatorilor "trial-and-error process", care implică încercarea alternativă viabilă întrucât nu există o soluție sigură. O altă soluție este cea a atingerii obiectivului prin aplicarea unei strategii de control. Aceasta este posibilă prin utilizarea acestei informații în scopul de a găsi o soluție dintre posibilitățile algoritmi de căutare.

6.1. Căutarea exhaustivă (Trial-and-error search)

Metoda cea mai simplă este cea de aplicare a operatorilor de căutare exhaustivă. Aceasta este o strategie de căutare exhaustivă care implică aplicarea tuturor operatorilor de căutare până la obținerea soluției.

Există două metode de aplicare a operatorilor de căutare exhaustivă: metoda de aplicare exhaustivă și metoda de aplicare selectivă. Metoda de aplicare exhaustivă implică aplicarea tuturor operatorilor de căutare până la obținerea soluției. Metoda de aplicare selectivă implică aplicarea operatorilor de căutare care sunt cei mai apropiați de soluție.

Metoda de aplicare exhaustivă este cea mai simplă și cea mai ușor de aplicat. Aceasta implică aplicarea tuturor operatorilor de căutare până la obținerea soluției. Metoda de aplicare selectivă este cea mai complexă și cea mai dificil de aplicat. Aceasta implică aplicarea operatorilor de căutare care sunt cei mai apropiați de soluție.

S-a arătat în capitolul precedent că o problemă reprezentată în spațiul stărilor conduce la o rezolvare ce se reduce la căutare în spațiul stărilor. Alegerea tehnicii de rezolvare este dependentă de tipul problemei, mai multe tehnici fiind posibil a fi aplicate. Cea mai simplă tehnică este cea de selecție a operatorilor "trial-and-error process", tehnica ce nu este totdeauna o alternativă viabilă întrucît nu oferă o căutare sistematică în spațiul stărilor. O altă soluție este cea a atingerii costului minim, iar în situația în care costul atingerii obiectivului poate fi predictat, timpul de căutare este corespunzător scurtat prin utilizarea acestei informații. În acest capitol se vor analiza o parte dintre posibili algoritmi ce căutare.

6.1. Căutarea exhaustivă (Trial-and-error searching)

Metoda cea mai simplă este cea de aplicare a tuturor operatorilor selectați pînă ce scopul este atins. Algoritmul asociat este prezent mai jos.

```
stabilește starea inițială
while (stare # starea obiectiv)
{
    selectează un operator aplicabil în această stare pe care îl
    aplică;
    stare = operator (stare); /* aplicarea operatorului la stare
    produce o altă stare ce devine starea actuală */
}
```

În pasul de selecție a operatorului, acesta este ales aleator dintre operatorii aplicabili. Procedura devine stochastică, însă nu se garantează atingerea obiectivului. Metoda simplă de acest tip este capabilă să soluționeze probleme simple. Pentru a justifica modul în care se ajunge la soluție schimbările succesive de stare sînt înregistrate în memorie.

6.2. Căutarea prin examinarea semantică a nodurilor

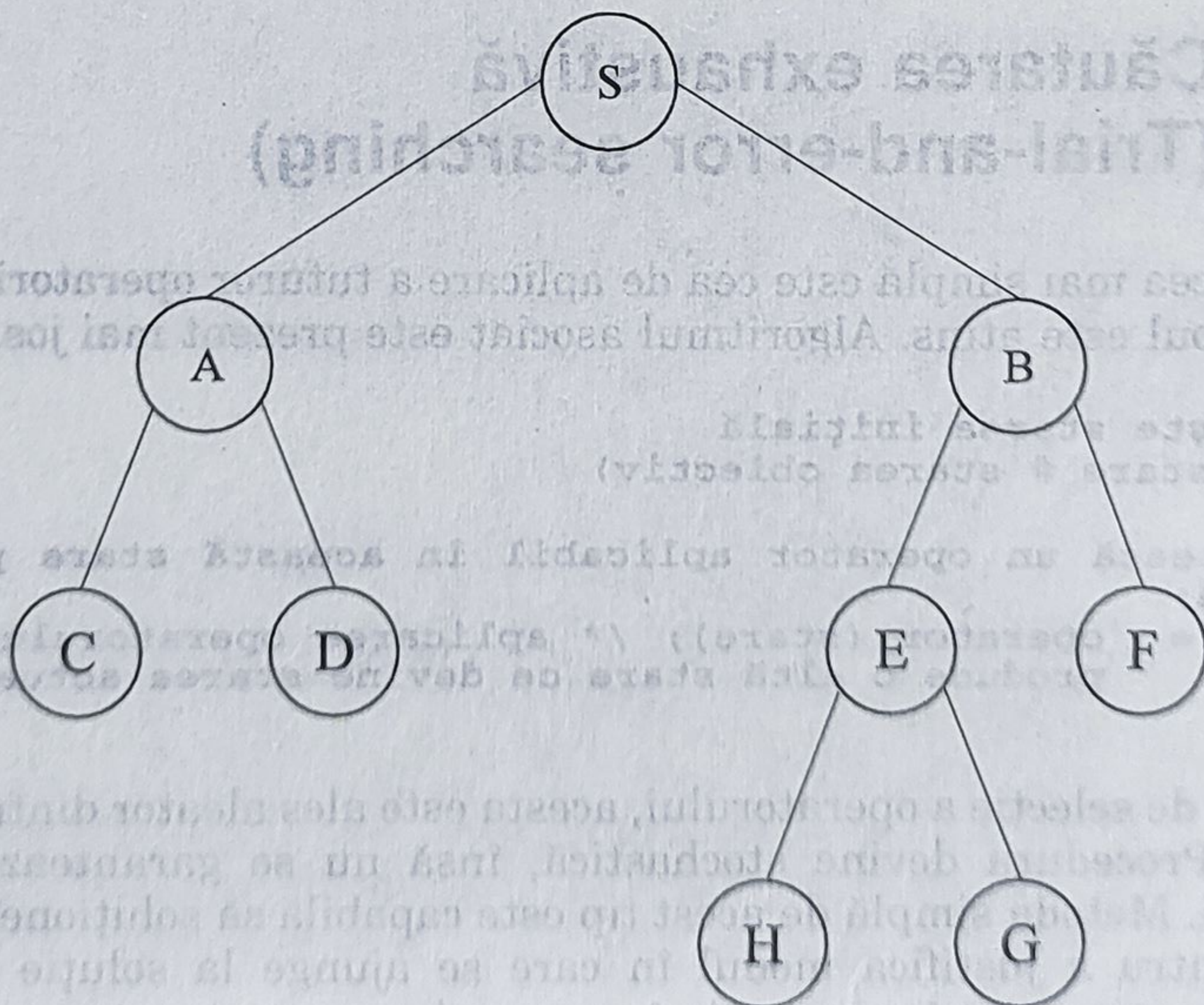
Această metodă permite examinarea spațiului stărilor într-o secvență predeterminată, pînă cînd obiectivul este atins. Metoda poate fi segmentată în căutare pe verticală și căutare pe orizontală, funcție de ordinea în care procesul avansează în spațiul stărilor.

6.2.1. Căutarea în adîncime

Căutarea în adîncime poate fi implementată pe calculator în mod direct prin utilizarea funcțiilor de înaintare și revenire (f , b). Acest tip de căutare este caracterizat prin faptul că pornește de la rădăcină și traversează graful de la un nivel la altul prin urmărirea arcelor. O secvență a ramurii distincte x_1, x_2, \dots, x_n este explorată cînd $b(x_1) = p$ (nod rădăcină), $b(x_2) = f(x_1), \dots, b(x_n) = f(x_{n-1})$ și nu există x_{n+1} în L astfel încît $b(x_{n+1}) = f(x_n)$, ceea ce arată că $f(x_n)$ este o frunză. Cînd nodul dorit nu este găsit o altă secvență de ramificații către noduri frunză sînt inițiate. Procesul continuă pînă cînd nodul dorit este localizat sau s-a ajuns la ultimul nod frunză.

Se consideră mai întîi căutarea cînd spațiul stărilor problemei este reprezentat printr-un arbore ca cel din fig. 6.1. Metoda de căutare verticală este cunoscută sub numele de căutare în adîncime (depth-first).

▼ Fig. 6.1. Exemplu de arbore cu arce reprezentînd pointeri



Căutarea pornește de la nodul S și continuă pînă cînd nodul obiectiv este atins. Dacă obiectivul este marcat prin G secvența de parcurgere este $\{A, C, D, B, E, H, G\}$. Dacă nodul n la care s-a ajuns nu este nodul obiectiv (în cazul de față G), și dacă nodul n are noduri fiu, acestea vor fi examinate mai întîi. Dacă nodul curent nu are noduri fiu, programul returnează controlul la nodul părinte m al lui n , și examinează următorul fiu ce pornește din m . Calea către nodul obiectiv poate fi obținută printr-o aranjare astfel încît nodul părinte al fiecărui nod fiu să fie reținut în vederea căutărilor ulterioare. Modalitatea de realizare presupune asocierea la fiecare nod a unui pointer către nodul său părinte. În figură pointerii sînt indicați prin săgeți. Cînd obiectivul este găsit calea de atingere a acestuia este determinată prin parcurgerea înapoi ghidată de secvența pointerilor.

Se dă mai jos algoritmul pentru căutarea nodurilor fiu ale nodului n , aplicînd toți operatorii valizi ce expandează nodul n . Normal, în rezolvarea problemei pentru generarea arborilor de căutare sînt explorate procedeele de expandare a nodurilor. Arborele este definit indirect prin reprezentarea spațiului stărilor. Nodurile ce au fost atinse dar nu au fost expandate sînt introduse într-o listă numită și "open_list".

```

stabilește nodul de start din listă;
while (1)
{
    if (open_list = lista_vidă) exit (fail);
    else
    {
        n = prim_nod (lista); /* aduce primul nod din lista */
    }
    if obiectiv (n) exit (succes); /* dacă n este obiectivul
    căutarea se termină cu succes */
    else
    {
        remove (n, open_list); /* nodul n este scos din open_list */
        expandează n, pune toate nodurile fiu în open_list și
        atașează pointeri de la nodurile fiu la nodurile părinte
        */
    }
}

```

Obiectiv (n) este o funcție logică ce întoarce valoarea TRUE dacă n este nod obiectiv și FALSE altfel. Dacă n nu are noduri fiu procesul continuă cu următorul nod din listă. Nodurile fiu sînt introduse în listă în orice ordine, însă este mult mai eficient ca nodul obiectiv să fie primul în listă față de alte noduri ce au același părinte. Pentru arborele din figură succesiunea nodurilor în listă este următoarea:

$$(S) \rightarrow (A, B) \rightarrow (C, D, B) \rightarrow (D, B) \rightarrow (B) \rightarrow (E, F) \rightarrow (H, G, F) \rightarrow (G, H)$$

Pe parcursul parcurgerii grafului orice nod ce a fost expandat va fi extras din listă. În scopul urmăririi raționamentului se pot dispune nodurile deja examinate într-o listă numită și closed_list, care la început este vidă. Dacă se utilizează și o astfel de listă algoritmul de lucru se modifică, structura sa fiind cea de mai jos.


```

stabilește nodul de start în listă;
while (1)
{
    if (open_list = lista_vidă) exit (fail);
    else
    {
        n = prim_nod (lista); /* aduce primul nod din listă */
    }
    if (obiectiv (n)) exit (succes); /* dacă n este obiectivul
        căutarea se termină cu succes */
    else
    {
        remove (n, open_list); /* nodul n este scos din open_list */
        add (n, closed_list); /* nodul n este adăugat în closed_list */
        expandează n, pentru a genera toate nodurile fiu. /* Orice nod
            ce apare în closed_list sau open_list va conține și pointeri
            la nodul sau părinte */
    }
}

```

La această procedură nodurile ce sînt adăugate în liste, prin expandare la open_list sau abandon la closed_list, vor fi plasate în vârful stivei încît la următorul pas sînt primele luate în considerație.

6.2.2. Căutarea pe orizontală (în lărgime)

La căutarea pe orizontală (breadth-first search), nodurile de la același nivel sînt parcurse mai întîi. În exemplul din fig. 6.1 ordinea de căutare va fi A, B, C, D, E, F, H, G. Nodurile sînt examinate în ordine începînd cu nodul de start după algoritmul

```

pune nodul de start în open_list;
while (1)
{
    if (open_list = lista_vidă) exit (fail);
    n = prim_nod (open_list);
    if (obiectiv (n)) exit (succes);
    remove (n, open_list);
    add (n, closed_list);
    expandează n, și generează toate nodurile fiu. Acelea dintre
    noduri care nu sînt în open_list sau closed_list vor fi puse în
    partea inferioară a open_list cu pointeri asociați la nodul n;
}

```

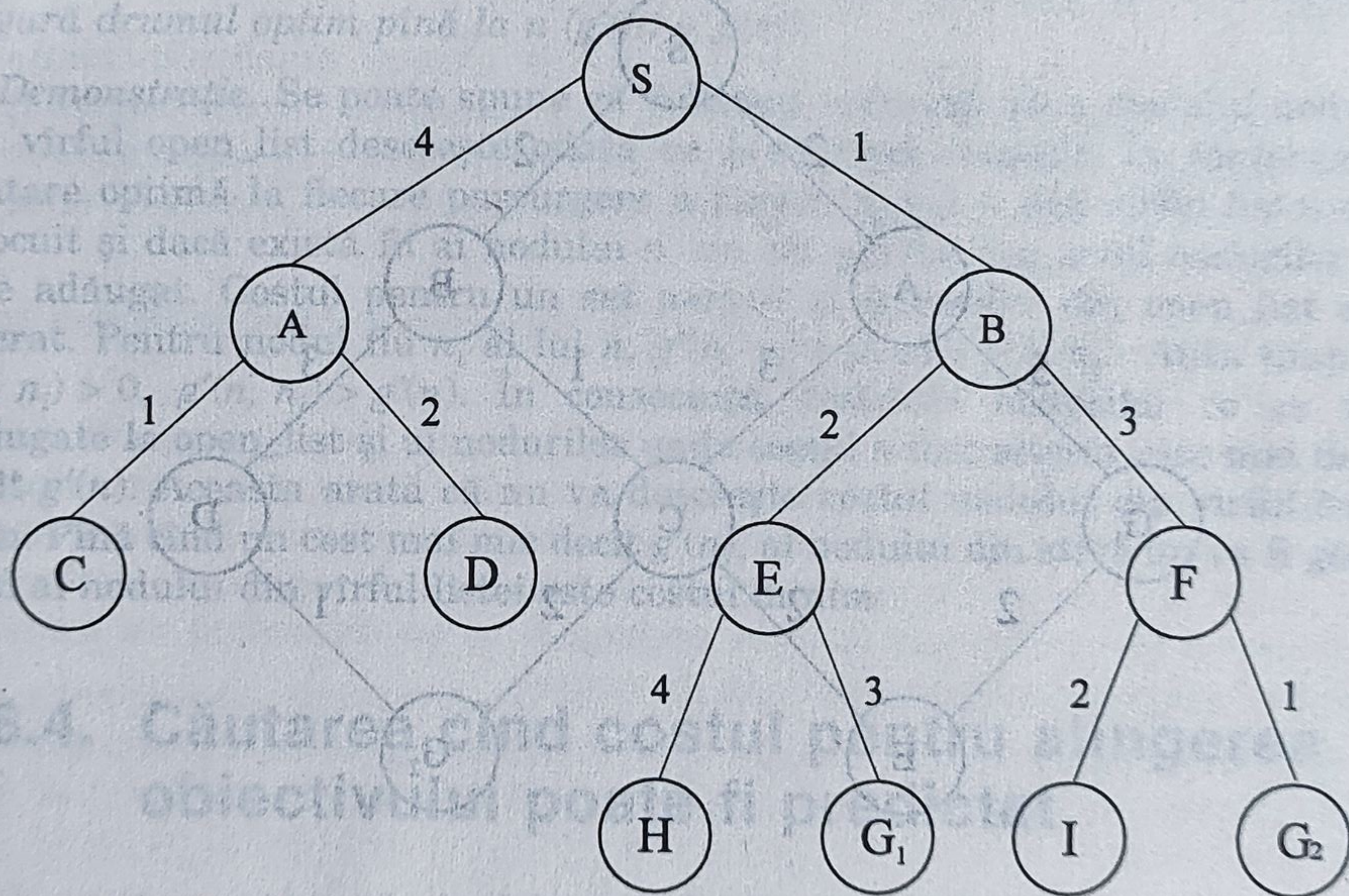
Procedura este similară cu cea de căutare în adîncime, excepție făcînd faptul că nodurile generate sînt adăugate la capătul inferior al listei și nu la cel superior. În general, dacă nodul obiectiv este în adîncimea arborelui, căutarea în adîncime este cea indicată, altfel se preferă căutarea în lărgime. Totuși, dacă adîncimea arborelui și numărul de ramuri ce pornesc de la un singur nod este mare se poate lua greu o decizie asupra metodei de căutare.

6.3. Căutarea soluției optime

La generarea spațiului stărilor este posibil a fi aplicați acei operatori ce au condițiile de aplicabilitate îndeplinite. Este normal ca eficientizarea căutării să fie făcută prin asocierea la fiecare operator a unei *funcții de valoare*, *funcții de cost* sau *funcții de distanță* ce oferă o măsură a consecințelor aplicării operatorului.

Se consideră problema găsirii costului minim în cazul în care spațiul stărilor este reprezentat printr-un graf și costul operatorilor corespunzători fiecărui arc este dat. Se presupune că toate costurile sînt pozitive. Fie ca exemplu arborele din fig. 6.2 în care nodurile obiectiv sînt G_1 și G_2 .

▼ Fig. 6.2. Arbore avînd costuri de traversare definite



Strategia de căutare este bazată pe examinarea mai întâi a acelor noduri care au un cost minim pornind de la nodul de start. În exemplul de mai sus nodurile A și B sînt generate prin expandarea nodului de start S . Întrucît atingerea nodului B implică cost minim pornind din S , se va examina B , care nu este nod obiectiv, avînd ca efect expandarea imediată în E și F . Cum atingerea lui E implică un cost mai mic, acesta este următorul nod examinat, expandarea sa ducînd la nodul G_1 .

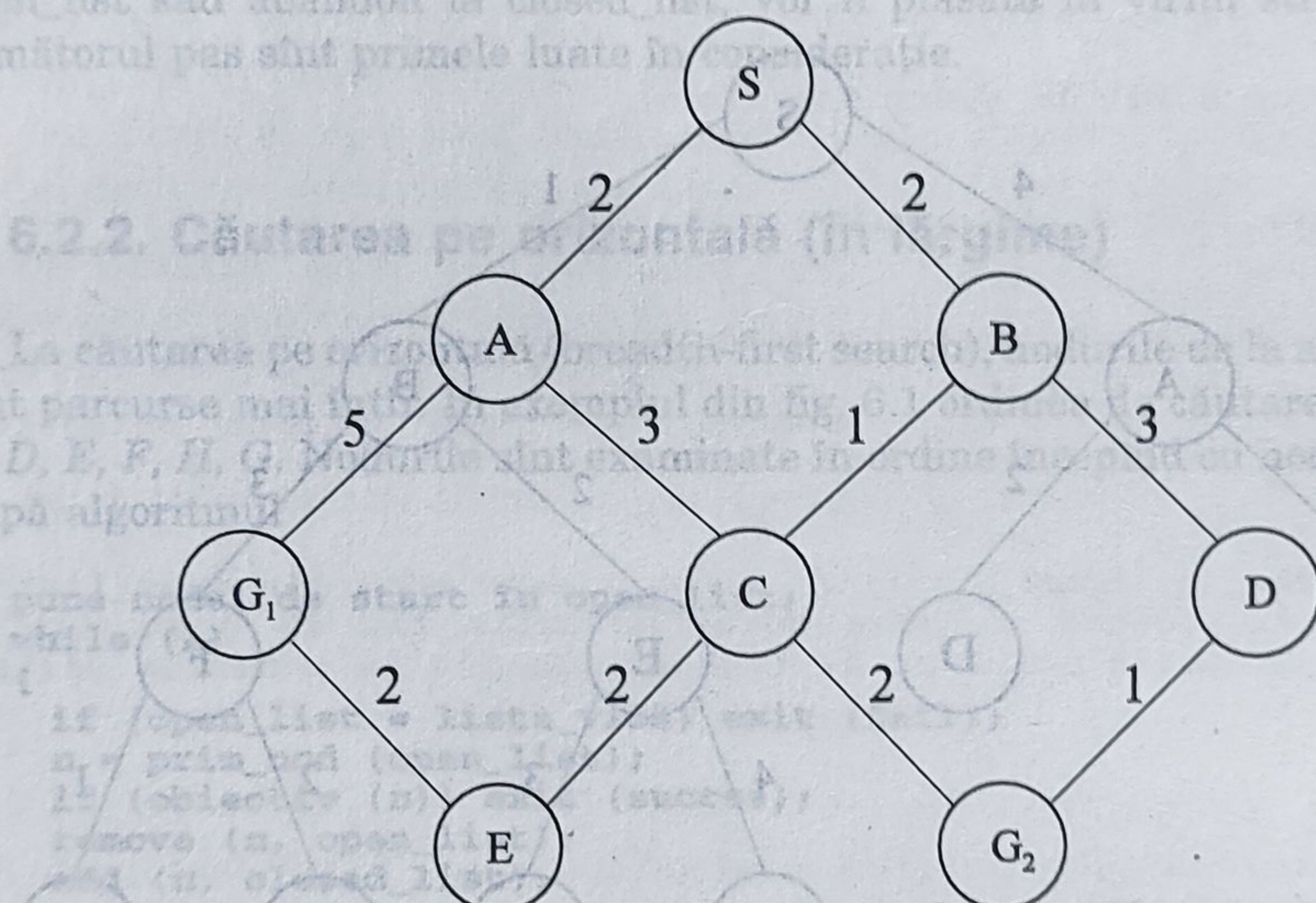
Se va reprezenta costul drumului optim de la nodul de start la nodul n prin $g(n)$, costul celei mai bune parcurgeri prin $g'(n)$, adică $g'(n)$ este valoarea inferențială a lui $g(n)$. Pentru o cale selectată $g(n) \geq g'(n)$. Algoritmul de căutare în arbori este:

```

pune nodul de plecare în open_list  $g'(n) = 0$ ;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
   $n = \text{prim\_nod}(\text{open\_list})$ ;
  if (obiectiv ( $n$ )) exit (succes);
  remove ( $n$ , open_list);
  expandează  $n$ , calculează  $g'(n)$  pentru nodurile fiu ale lui  $n$ ,
  atașează pointeri de la  $n_i$  la  $n$ . Depune toate nodurile fiu în
  open_list și aranjează nodurile în această listă în ordinea
  costului minim.
}

```

▼ Fig. 6.3. Graph în care costul este definit



Aplicarea algoritmului de mai sus pentru graful din fig. 6.3 determină succesiunea de noduri:

$S(0) \rightarrow (A(2)B(3)) \rightarrow (B(3)C(5)G_1(7)) \rightarrow (C(4)D(6)G_1(7)) \rightarrow (G_2(6)D(6)E(6)G_1(7))$

Se va arăta faptul că dacă graful este finit există un drum optim de la nodul de start la nodul obiectiv, prin aplicarea următoarelor teoreme. Soluția se obține prin procedura optimală de căutare.

Teorema 6.1. Într-un graf finit există un optim de la nodul de start la nodul obiectiv pe care procedura îl găsește.

Demonstrație. Se va presupune că algoritmul de căutare nu se termină cu succes. În acest caz `open_list` este vidă, `closed_list` conține toate nodurile explorate pînă în acel moment. Deoarece graful are o soluție se va presupune că drumul conține nodurile (n_0, n_1, \dots, n_m) , în care n_0 este identic cu nodul S și n_m este identic cu nodul obiectiv. Parcurgînd drumul indicat înapoi se găsește nodul n_i care a fost introdus în `closed_list` iar nodul n_{i+1} nu se găsește în `closed_list`. Întrucît n_i este inclus în `closed_list` el va fi expandat. Ca urmare a expandării n_{i+1} va fi pus mai întîi în `open_list`. Ca urmare n_{i+1} trebuie să fie examinat înainte ca `open_list` să devină vidă. Dacă n_{i+1} este un nod obiectiv căutarea duce la succes. Dacă n_{i+1} nu este un nod obiectiv este pus în `closed_list`. În ambele cazuri presupunerea de eșec este contrazisă. Întrucît graful este finit și în urma căutării nu se ajunge la eșec atunci trebuie să se obțină un succes.

Rațiunea pentru care un nod fiu n_i ce se găsește în `closed_list` nu este și în `open_list` pornește de la faptul că $g'(n, n_i)$ nu poate fi mai mic decît $g'(n_i)$, așa că o cale trecînd prin n nu poate fi considerată un candidat la soluție, așa cum se demonstrează în teorema următoare.

Teorema 6.2. *La momentul în care nodul n este expandat căutarea optimă asigură drumul optim pînă la n ($g'(n) = g(n)$).*

Demonstrație. Se poate spune că valoarea inferențiată a costului nodului din vîrfurile `open_list` descrește odată cu avansarea căutării. În secvența de căutare optimă la fiecare parcurgere a buclei, nodul n din vîrfurile listei este înlocuit și dacă există fii ai nodului n , un set parțial din setul nodurilor fiu este adăugat. Costul pentru un set parțial al nodurilor din `open_list` este alterat. Pentru nodul fiu n_i al lui n , $g'(n, n_i) = g'(n) + c(n, n_i)$. Atîta timp cît $c(n, n_i) > 0$, $g'(n, n_i) > g'(n)$. În consecință, costurile nodurilor ce au fost adăugate la `open_list` și al nodurilor unde costul a fost alterat este mai mare decît $g'(n)$. Aceasta arată că nu va descrește costul nodului din vîrfurile listei `open`. Pînă cînd un cost mai mic decît $g'(n)$, al nodului din stivă nu va fi găsit, $g'(n)$ al nodului din vîrfurile listei este costul minim.

6.4. Căutarea cînd costul pentru atingerea obiectivului poate fi predictat

La acest moment principalele metode de căutare grafică au fost expuse. Cum problema are o reprezentare în spațiul stărilor, căutarea poate fi făcută mai eficient utilizînd informațiile pentru reprezentarea problemei. De exemplu, în problema labirintului dacă se cunoaște poziția obiectivului relativ la punctul de plecare, se va selecta calea ce încarcă cel mai puțin posibil costul. La problema careului cu opt numere, o informație a apropierei de obiectiv este dată de poziția punctelor față de starea obiectiv. Informația rezultată din predicția costului față de obiectiv, chiar dacă nu este suficient de precisă poate fi utilizată. Buclele moarte pot determina creșterea timpului (și implicit

a costului) de atingerea obiectivului. Astfel de cunoștințe sînt de natură euristică, întrucît nu garantează corectitudinea completă. Căutarea bazată pe cunoștințe euristice se numește și căutare euristică. Se cunosc mai multe tehnici de căutare euristică, tehnici ce se vor expune mai jos.

6.4.1. Metoda gradientului (Hill-climbing method)

Ideea de bază a căutării în cazul unei strategii irevocabile este cea prin care pornind de la starea inițială să se minimizeze distanța ce separă starea curentă obținută, de starea obiectiv a problemei. Metodele matematice cunoscute în teoria sistemelor găsesc un bun teren de aplicabilitate în această situație. O metodă des utilizată este metoda gradientului (metoda hill climbing) ce se utilizează în procesele de căutare multidimensională a extremelor funcțiilor de mai multe variabile sau a soluțiilor sistemelor de ecuații neliniare. O serie de probleme duc la complexitate în procesul generării stărilor ce se reflectă în caracterul neterminal al stărilor obținute. Aceasta nu permite testarea concordanței cu obiectivul pentru toate stările posibil de obținut dintr-un nod dat. Generarea spațiului stărilor se desfășoară după strategii ce urmăresc două criterii de optimalitate:

- ♦ *optimalitate locală* - avînd ca obiectiv găsirea numărului minim de pași ce duc la o stare terminală, fără epuizarea tuturor combinațiilor posibile. Starea precedentă este marcată pentru a se asigura revenirea ulterioară în caz de eșec și reluarea procesului prin generarea unui nou element terminal;
- ♦ *optimalitatea globală* - se referă la obținerea soluției problemei cu un număr minim de reveniri.

Strategiile de control irevocabile se bazează pe cunoașterea locală, determină operatorul aplicat de la un pas la altul, noua stare reprezintă un context necunoscut, context folosit pentru determinarea unui alt operator pînă se atinge obiectivul. Se poate spune că secvența de operatori definește o cunoaștere globală despre soluția problemei ce se determină ca urmare a aplicării operatorilor prin strategia de control.

Ideea de bază a acestei metode se bazează pe faptul că direcția celei mai mari creșteri a funcției $f(x_1, x_2, \dots, x_n)$ este dată de gradientul funcției.

$$\text{grad } f(x_1, x_2, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right).$$

Aplicarea metodei este dificilă dacă numărul variabilelor este mare, numărul funcțiilor de cost este mare, funcția are mai multe valori extreme.

La căutarea într-un graf, strategia traversării pentru atingerea obiectivului prin alegerea acelor noduri care sînt predictate a fi apropiate de obiectiv este numită metoda de gradient. Specific pentru alegerea următorului nod după n , se calculează $h(n_i)$ pentru toate nodurile fiu (n_1, n_2, \dots, n_m) ale lui n și stabilește ca următor nod acel nod pentru care $h(n_i)$ este mai mic. Funcția $h(n)$ este o măsură a apropierii de obiectiv. Algoritmul de căutare este:


```

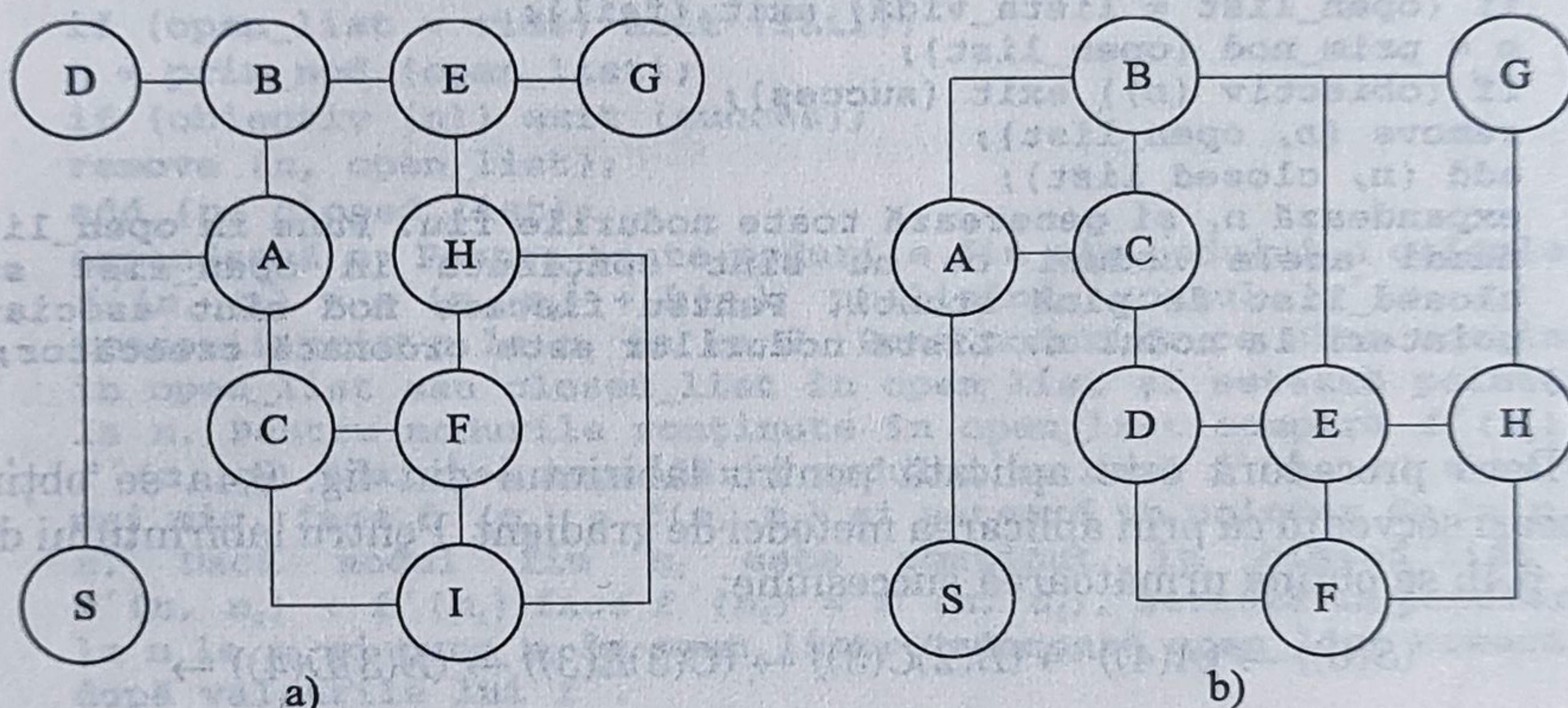
n = nodul de start;
while (1)
{
    if (obiectiv (n)) exit (succes);
    expandează n, calculează  $h(n_i)$  pentru toate nodurile fiu  $n_i$  și
    face nodul fiu nod următor (next_n) acel nod care are valoarea
    minimă;
    if ( $h(n) < h(next\_n)$ ) exit (fail);
    n = next_n;
}

```

Se poate imagina obiectivul ca vârful unui munte și $h(n)$ ca diferență de înălțime între nodul n și vîrf. Metoda urmărește apropierea de obiectiv prin gradient maxim. Dacă există numai un vîrf metoda permite atingerea obiectivului, însă pot exista vîrfuri locale ce sînt atinse, ce nu permit apoi apropierea de maximul global.

▼ Fig. 6.4. Exemple de căutare utilizînd metode de gradient

- a) caz în care se obține succes
b) caz în care se ajunge la eșec



Un exemplu de problemă la care metoda este aplicabilă este cea a traversării labirintului, ilustrată în fig. 6.4. Pentru cazul din fig. 6.4a obiectivul se atinge, iar pentru cel din fig. 6.4b se obține eșec. În ambele cazuri, costul de la poziția caracteristică nodului n la obiectiv, este calculat cu formula:

$$h(n) = | \text{coordonata } x \text{ a obiectivului} - \text{coordonata } x \text{ a nodului } n | + | \text{coordonata } y \text{ a obiectivului} - \text{coordonata } y \text{ a nodului } n |$$

În cazul fig. 6.4a obiectivul poate fi atins prin secvența de noduri $S \rightarrow A \rightarrow B \rightarrow E \rightarrow G$, care este un drum optim ce restrînge aria de căutare. În cazul din fig. 6.4b se ajunge la eșec după parcurgerea secvenței $S \rightarrow A \rightarrow C \rightarrow B$, în care B reprezintă un vîrf dar care nu este un obiectiv. Pentru a ajunge la obiectiv ar trebui atins punctul E ce determină o depărtare de obiectiv.

6.4.2. Metoda celei mai bune prime căutări

În metoda gradientului maxim dacă sînt și alte extreme locale (valori minime ale lui $h(n)$) decît obiectivul nu se poate garanta continuarea algoritmului pentru atingerea acestuia. Tehnica cunoscută sub numele de metoda celei mai bune prime căutări se bazează pe luarea deciziei după examinarea spațiului stărilor. Toate nodurile ce au fost examinate prin această tehnică pînă la momentul curent sînt stocate, urmînd ca nodul cel mai apropiat de obiectiv să fie expandat. Pînă aici metoda este similară cu cea bazată pe gradient, însă atunci cînd $h(n)$ devine constant sau acțiunile permise în starea curentă produc depărtarea de obiectiv, un alt nod din listă va fi expandat. Algoritmul pentru căutarea în grafuri după această metodă este dat mai jos.

```
pune nodul de start  $S$  în open_list;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
   $n$  = prim_nod (open_list);
  if (obiectiv ( $n$ )) exit (succes);
  remove ( $n$ , open_list);
  add ( $n$ , closed_list);
  expandează  $n$ , și generează toate nodurile fiu. Pune în open_list
  numai acele noduri ce nu sînt conținute în open_list sau
  closed_list de pînă atunci. Pentru fiecare nod sînt asociați
  pointeri la nodul  $n$ . Lista nodurilor este ordonată crescător;
}
```

Dacă procedura este aplicată pentru labirintul din fig. 6.4a se obține aceeași secvență ca prin aplicarea metodei de gradient. Pentru labirintului din fig. 6.4b se obține următoarea succesiune:

$$(S(6)) \rightarrow (A(4)) \rightarrow (B(2)C(3)) \rightarrow (C(3)E(3)) \rightarrow (E(3)D(4)) \rightarrow \\ \rightarrow (H(2)D(4)) \rightarrow (G(0)D(4)F(4))$$

Pentru utilizarea metodei celei mai bune prime căutări este necesar să se cunoască costul atingerii obiectivului, adică o serie de cunoștințe despre problemă.

6.4.3. Soluția optimă cînd costul poate fi predictat

În paragrafele anterioare s-au descris metode de căutarea soluției optime fără utilizarea cunoștințelor despre problemă. Se încearcă descrierea unei noi metode de găsire a soluției optime cînd costul ajungerii la scop poate fi predictat. Pentru un nod arbitrar n al grafului, se va nota cu $g(n)$ costul drumului optim de la nodul inițial S la nodul n , și $h(n)$ costul drumului optim de la nodul n la nodul obiectiv. Dacă între noduri nu există drum aceste costuri sînt infinite. Costul drumului optim de la nodul inițial la obiectiv este dat prin următoarea formulă:

$$f(n) = g(n) + h(n)$$

și problema este de a găsi drumul optim (partea care dă $f(S)$) de la nodul de start. Dacă $f(n)$ este cunoscută exact, soluția poate fi obținută prin înlanțuirea nodurilor cu cel mai mic $f(n)$ ce pornesc de la S . În practică fie $g(n)$ fie $h(n)$ nu sînt cunoscute cu precizie, așa că este necesară căutarea.

Ca și în exemplele precedente, valoarea costului inferențiat $h(n)$ către obiectiv este dat pentru toate nodurile. Se va nota cu $g'(n)$ valoarea lui $g(n)$ pentru nodurile deja generate ce au cost minim și conțin un drum ce trece prin n . Valoarea inferențiată $f'(n)$ a costului drumului optim trecînd prin n este dată de formula

$$f'(n) = g'(n) + h'(n)$$

Strategia de căutare utilizînd această funcție de evaluare este cunoscută sub numele de algoritmul A și este prezentat mai jos.

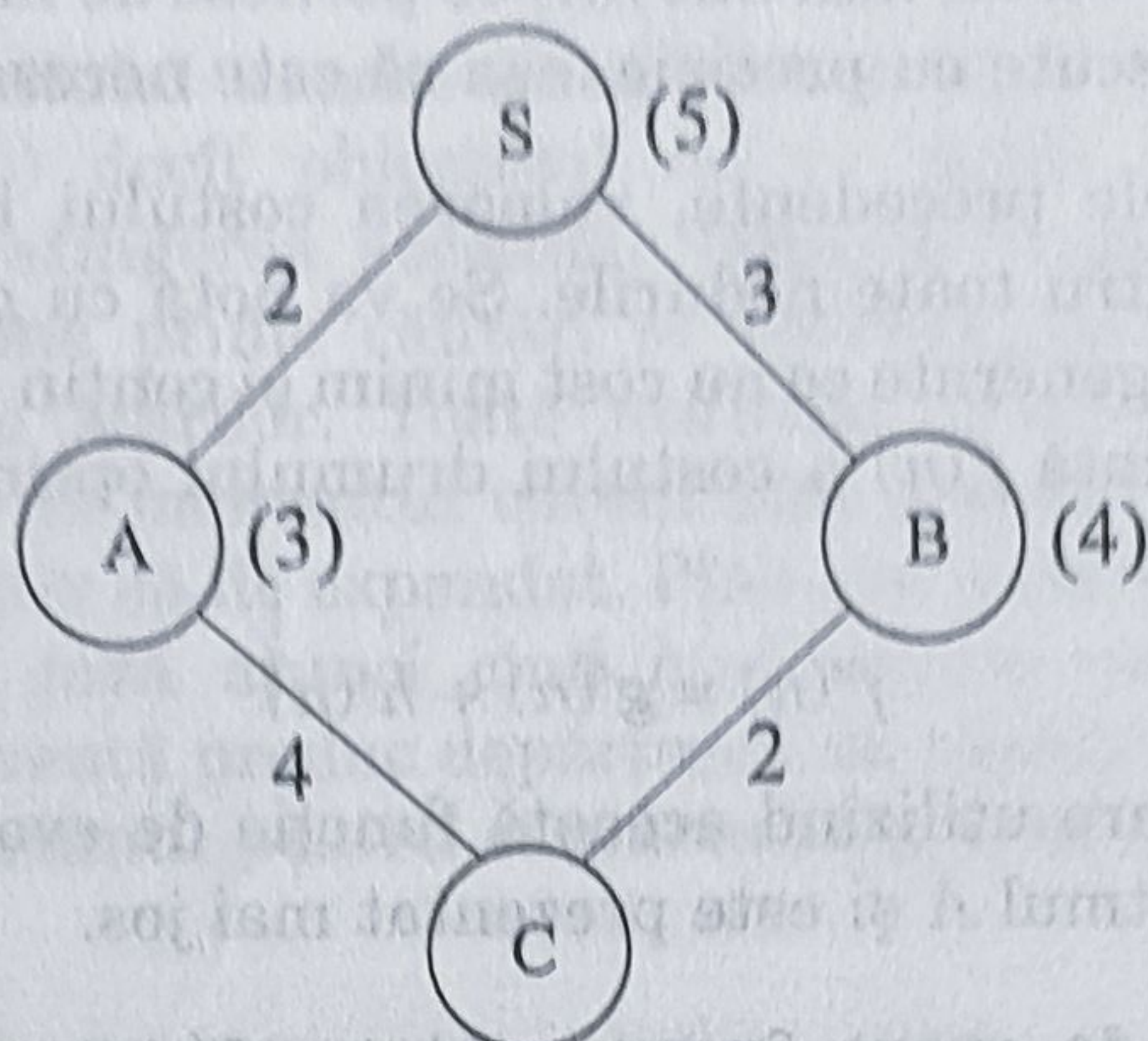
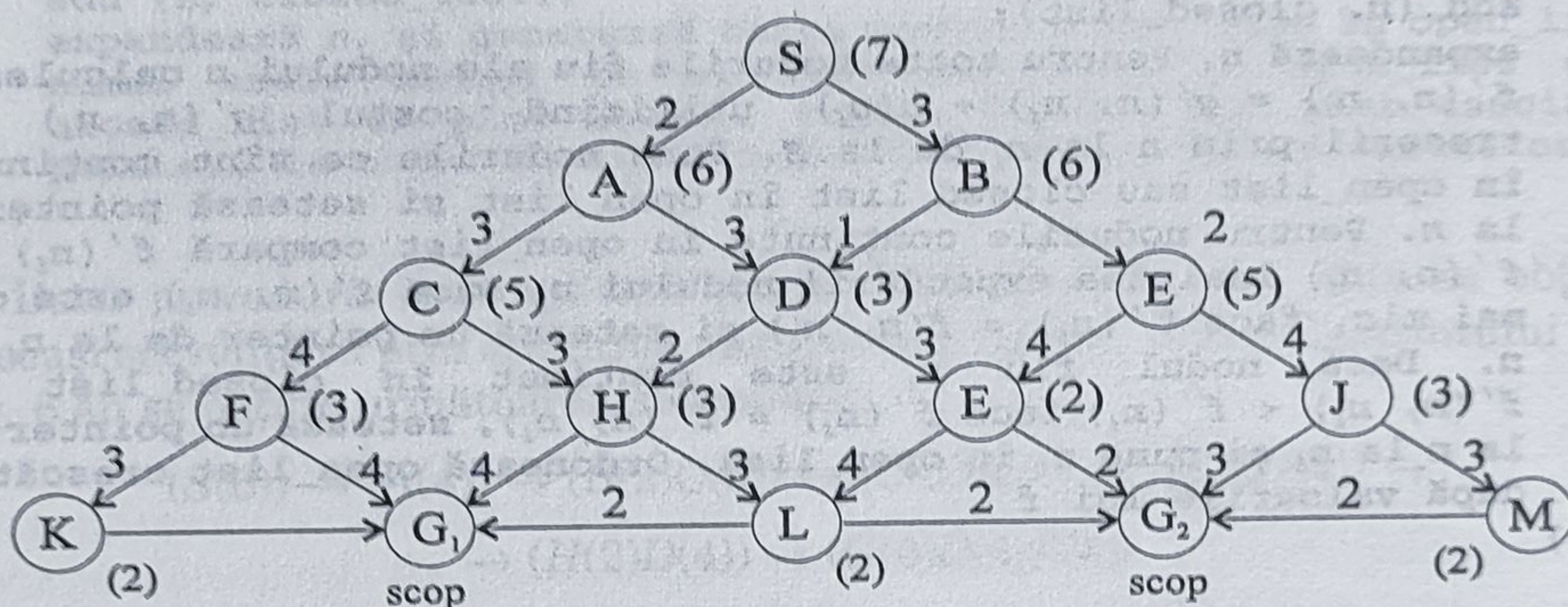
```

Introduce nodul de start în open_list;  $f'(S) = h'(S)$ ;
While (1)
{
  if (open_list = vidă) exit (fail);
   $n = \text{prim\_nod}(\text{open\_list})$ ;
  if (obiectiv ( $n$ )) exit (succes);
  remove ( $n$ , open_list);
  add ( $n$ , closed_list);
  expandează  $n$ . Pentru toate nodurile fiu ale nodului  $n$  calculează
   $f'(n, n_i) = g'(n, n_i) + h(n_i)$  utilizînd costul  $g'(n, n_i)$  al
  trecerii prin  $n$  la  $n_i$  de la  $S$ . Pune nodurile ce sînt conținute
  în open_list sau closed_list în open_list și setează pointerii
  la  $n$ . Pentru nodurile conținute în open_list compară  $f'(n_i)$  și
   $f'(n, n_i)$  înaintea expandării nodului  $n$ . Dacă  $f'(n, n_i)$  este cel
  mai mic, face  $f'(n_i) = f(n, n_i)$  și setează un pointer de la  $n_i$  la
   $n$ . Dacă nodul fiu  $n_i$  este conținut în closed_list și
   $f'(n, n_i) < f'(n_i)$  face  $f'(n_i) = f'(n, n_i)$ , setează un pointer de
  la  $n$  la  $n_i$  și pune  $n_i$  în open_list. Ordonează open_list crescător
  după valorile lui  $f'$ .
}

```

Dacă se consideră $h(n) = 0$ algoritmul are aceeași comportare cu algoritmul de căutare optimă. Dacă graful este finit, se poate arăta prin aceeași metodă bazată pe teorema 6.1, că drumul optim este găsit atunci cînd un nod de start și un nod obiectiv există. Totuși, nu se poate garanta că algoritmul va obține soluția optimă. În exemplul din fig. 6.5 sînt ilustrate în parantezele asociate nodurilor valorile funcției $h(n)$ pentru nodul n . Mai întîi se va expanda nodul S pentru a se obține nodurile sale fiu A și B . Se calculează funcțiile respective de evaluare $f'(A) = 2 + 3$ și $f'(B) = 3 + 4$, open_list este (A, B). Cînd se citește A din open_list și este expandat se va obține nodul G . $f'(G) = 6 + 0$ așa că open_list devine (G, B). Cînd este preluat G căutarea se termină. Soluția este drumul $S \rightarrow A \rightarrow G$. Cu toate acestea calea nu este cea de cost minim. Rațiunea pentru care drumul de cost minim $S \rightarrow B \rightarrow G$ nu este găsit este dată de faptul că $f(B)$ devine mai mare decît $f'(B)$.

▼ Fig. 6.5. Graf în care algoritmul A nu găsește soluția optimă

▼ Fig. 6.6. Graf în care condiția $h'(n) \leq h(n)$ este îndeplinită

Se poate dovedi că dacă $h'(n)$ excede $h(n)$, algoritmul va găsi soluția optimă. Dacă costul inferențiat este marginea inferioară a costului real ($h'(n) \leq h(n)$) algoritmul A se transformă în algoritmul A*.

În continuare se dă un nou exemplu de căutare într-un graf utilizând algoritmul A*. Graful în care se execută căutarea este prezentat în fig. 6.6. Ca și în fig. 6.5 valoarea lui $h'(n)$ este dată în paranteze asociate nodului. Schimbările în open_list obținute prin utilizarea algoritmului A* sînt marcate de indicarea lui $f'(n)$ în parantezele asociate nodurilor.

Se va obține secvența de noduri în open_list:

$(S(7)) \rightarrow (A(8)B(9)) \rightarrow (D(8)B(9)C(10)) \rightarrow (B(9)C(10)H(10)I(10))$
 $\rightarrow (D(7)C(10)E(10)H(10)I(10)) \rightarrow (H(9)I(9)C(10)E(10))$
 $\rightarrow (I(9)G_1(10)C(10)E(10)L(11)) \rightarrow (G_2(9)G_1(10)C(10)E(10)L(11))$

Procedura se termină cînd G_2 este extras din open_list. Soluția $S \rightarrow B \rightarrow D \rightarrow I \rightarrow G_2$ este soluția optimă ($h(S) = 9$). Este remarcabil faptul că un nod n care satisface $h'(n) \leq h(S)$ trebuie să existe în open_list și cel puțin pentru nodurile din calea optimă $h'(n) \leq h(S)$. În procesul de căutare nodul D este introdus în closed_list, în schimb atunci cînd se reîntoarce la D și găsește drumul de cost minim acesta este pus din nou în open_list. Pentru că $h' = 0$ este sigur marginea inferioară pentru h , căutarea optimă utilizînd această procedură este un caz particular al algoritmului A^* . Algoritmul A^* corespunde căutării optime fără utilizarea cunoștințelor euristice.

6.4.4. Caracteristici ale algoritmului A^*

Întrucît algoritmul A^* este un caz particular al algoritmului A , posedă toate caracteristicile acestuia din urmă. Dacă graful este finit și există un drum de la nodul de start la nodul obiectiv, algoritmul se termină prin găsirea unui drum între nodul de start și nodul obiectiv. Se arată mai jos că și în cazul în care graful este infinit algoritmul găsește soluția optimă.

Lema 6.1. *Reprezentînd nodul de start prin S , înaintea terminării algoritmului A^* , va putea exista un nod n care satisface relația $f'(n) \leq f(S)$.*

Demonstrație. Drumul optim de la S la nodul obiectiv este reprezentat printr-o serie (n_0, n_1, \dots, n_m) de noduri ($n_0 = S$, $n_m = \text{nod obiectiv}$). Inițial S se găsește în open_list și algoritmul nu s-a terminat încă, așa că este un nod al drumului optim dispus în open_list. Referitor la nodurile din open_list, notînd cu n nodul care apare primul în secvența nodurilor drumului optim, nodurile anterioare acestuia sînt conținute în closed_list, așa că drumul optim de la nodurile părinte lui n a fost găsit. Întrucît n aparține drumului optim de la S la obiectiv, drumul optim ce trece prin n a fost găsit. De aceea

$$f'(n) = g'(n) + h'(n) = g(n) + h'(n)$$

Întrucît în algoritmul A^* $h'(n) \leq h(n)$

$$f'(n) \leq g(n) + h(n) = f(n)$$

și deci n aparține drumului optim, așa că $f'(n) \leq f(S)$. De aici rezultă că există un nod n care satisface $f(n) = f(S)$.

Lema 6.2. *Dacă graful conține un drum de la nodul de start la nodul obiectiv, drumul către nodul obiectiv poate fi găsit prin algoritmul A^* .*

Demonstrație. S-a arătat pînă în prezent că dacă algoritmul A^* se termină, el poate găsi calea către nodul obiectiv. Se presupune că algoritmul A^* nu se termină și că lungimea arcului este reprezentată prin 1. Valoarea minimă a lungimii drumului de la nodul de start la nodul n este notată cu $d(n)$. Cum numărul ramurilor de la un singur nod este finit, continuarea căutării determină $d(n)$ pentru orice nod n din open_list ce devine mai mare decît un număr arbitrar M . Întrucît costul este pozitiv făcînd valoarea minimă m se obține $g(n) \geq d(n)m$. Întrucît, $g'(n) \geq g(n)$ și $f'(n) \geq g'(n)$,

$f'(n) \geq d(n)m$. Dacă se alege $M = f(S)/m$, căutarea produce $d(n)$ ce devine mai mare decât M , așa că se obține $f'(n) \geq f(S)d(n)/M > f(S)$. Aceasta contrazice Lema 6.1, deci algoritmul se termină.

Teorema 6.3. *Dacă graful conține un drum de la nodul de start la nodul obiectiv algoritmul A^* se termină prin găsirea soluției optime.*

Demonstrație. Se cunoaște din lema 6.2 ca algoritmul A^* se termină. S-a arătat de asemenea că dacă algoritmul se termină, drumul spre nodul obiectiv este găsit, precum și faptul că terminarea algoritmului cu găsirea nodului obiectiv asigură soluția optimă. Dacă se presupune că aceasta nu este soluția optimă, atunci $f(G) > f(S)$. Din lema 6.1, înainte de terminarea căutării există un nod n în `open_list` care satisface $f'(n) \leq f(S)$. Întrucât pentru nodul n , $f'(n) < f'(G)$ la acest moment, algoritmul A^* va selecta n și nu G . Aceasta contrazice presupunerea că se termină căutarea prin găsirea lui G .

Corolar 6.1. *$f'(n')$ pentru nodul n' expandat prin algoritmul A^* este mai mare decât costul soluției optime.*

Demonstrație. Nodul n' ce este expandat nu este un nod obiectiv și căutarea nu s-a terminat. Din lema 6.1 `open_list` conține un nod n care satisface $f'(n) \leq f(S)$. Algoritmul preia nodul pentru care f este minim din `open_list` și îl expandează ($f'(n') \leq f'(n)$). De aceea $f'(n') \leq f(S)$. Din teorema 6.3 se poate vedea că atunci $h'(n) = 0$, așa că algoritmul de căutare optimă permite găsirea soluției optime.

Se poate acum examina relația între funcția euristică $f'(n)$ și algoritmul A^* . Dacă $f'(n) = f(n)$, soluția poate fi găsită fără expandarea nodurilor ce nu sînt necesare. Se poate spera că apropierea lui $f(n)$ de $f'(n)$ va scădea numărul nodurilor ce sînt expandate. Presupunînd că se utilizează algoritmul A^* cu două funcții diferite de evaluare $f'_1(n)$ și $f'_2(n)$ (A_1 și A_2), dacă pentru toate nodurile n , $h_1(n) > h_2(n)$, se spune că gradul de cunoaștere pentru A_1 este mai mare decât cel pentru A_2 .

Teorema 6.4. *Fie doi algoritmi A^* , A_1 și A_2 , presupunînd că gradul de cunoaștere pentru A_1 este mai mare decât cel pentru A_2 nodurile ce sînt expandate prin A_1 vor putea fi expandate și prin A_2 .*

Demonstrație. Se va utiliza pentru demonstrație metoda inducției matematice. Teorema reține punctul final al căii de lungime 0 de la nodul de start. (Dacă S este un nod obiectiv nici A_1 , nici A_2 nu va determina expandarea, altfel ambele sînt expandate). Se va arăta că dacă teorema reține nodul pentru punctul de terminare al tuturor drumurilor de lungime k de la S , reține de asemenea toate punctele de terminare ale căilor de lungime $k - 1$. Se presupune că acesta este un nod expandat prin A_1 dar neexpandat prin A_2 . Denumind acest nod n , întrucât A_2 expandează nodul părinte, cînd A_2 se termină, n este scos din `open_list`. Cum n nu este selectat, cînd algoritmul se termină, se rețin următoarele relații:

$$f'(n) \geq f(S)$$

$$\text{Însă } f'(n) = g'_2(n) + h'_2(n),$$

$$h'_2(n) \geq f(S) - g'_2(n).$$

Pe de altă parte, întrucît A_1 a expandat n la acest moment

$$f'_1(n) \leq f(S), \text{ și}$$

$$f'_1(n) = g'_1(n) + h'_1(n) \text{ se obține}$$

$$h'_1(n) \leq f(S) - g'_1(n).$$

Întrucît orice nod care este expandat prin A_1 ce este nod în calea de lungime k va fi expandat și prin A_2 :

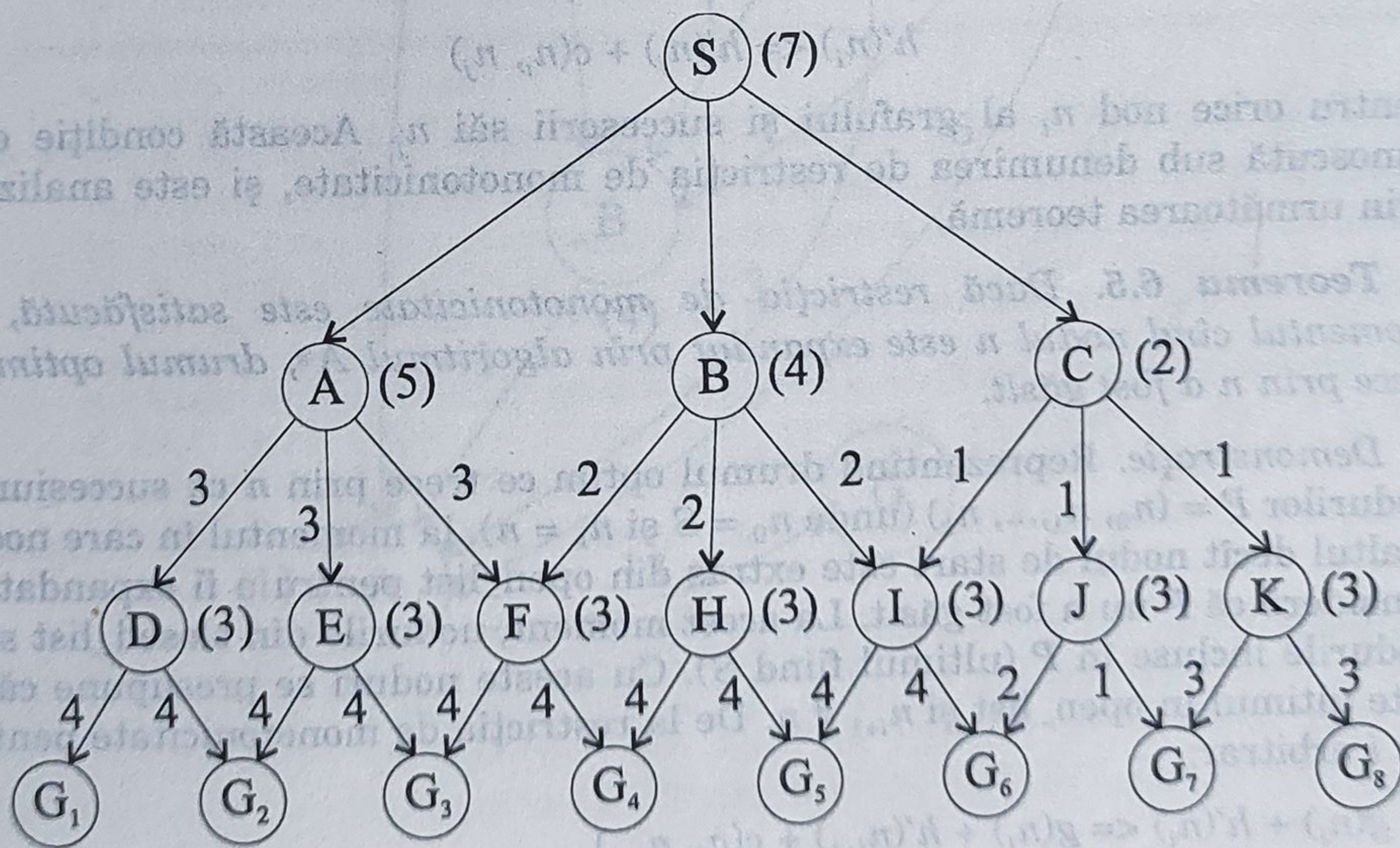
$$g'_2(n) \leq g'_1(n), \text{ și}$$

$$h'_1(n) \leq f(S) - g'_1(n) \leq f(S) - g'_2(n) \leq h'_2(n).$$

Aceasta contravine presupunerii că $h'_1 > h'_2$. Deci presupunerea că n există nu este adevărată.

De la această teoremă se poate vedea că algoritmul A^* utilizînd cunoașterea euristică $h(n)$ expandează mai puține noduri decît căutarea pentru soluția optimă fără utilizarea lui h' ($h' = 0$). În fig. 6.7 se indică un exemplul al utilizării algoritmului. Indiferent de tipul algoritmului de căutare se găsește soluția optimă ($S C J G_7$) avînd costul asociat 8. Atunci cînd $h' = 0$ toate nodurile către obiectivele G_1 la G_8 sînt expandate. Aceasta corespunde găsirii valorii minime a costurilor după găsirea tuturor drumurilor la obiectiv. În contrast, dacă valorile indicate în paranteze în fig. 6.7 sînt utilizate pentru funcția h , nu vor fi expandate decît acele noduri ce conduc la soluția optimă.

▼ Fig. 6.7. Exemplu indicînd efectul funcției euristice $h()$



6.4.5. Îmbunătățiri ale algoritmului A*

Numărul de noduri expandate este o măsură a eficienței de căutare. Așa cum se observă din algoritmul prezentat, nodurile ce au fost expandate sînt trecute în `closed_list` și pot fi readuse în `open_list`. Altfel spus, este posibil ca același nod să fie expandat de mai multe ori, la momente de timp diferite. Cu cît numărul de noduri ce se expandează scade, eficiența căutării crește. În fig. 6.8 se arată un exemplu de graf în care algoritmul A* expandează același nod de mai multe ori.

Operațiile de expandare a nodurilor utilizînd acest algoritm sînt arătate în tabela 6.1. Fiecare rînd din tabelă indică nodurile din `open_list` și `closed_list`, precum și costul inferențiat corespunzător f' . Pentru a indica faptul că un nod este în `closed_list` în tabelă s-a marcat prin simbolul *. Inițial, numai nodul de start S este în `open_list`, așa că $f'(S)$ este dată în primul rînd al tablei.

Apoi, S este pus în `closed_list` și nodurile A , B , C și D care sînt obținute prin expandarea lui S sînt dispuse în `open_list`. În continuare, nodul A pentru care f' este un minim este extras din `open_list` și depus în `closed_list` odată cu fiii săi. Cînd procedura este repetată nodul se va expanda în total de 17 ori. Dacă numărul de noduri este N , se cunoaște că numărul maxim de expandări ale nodurilor este 2^N . În opoziție, dacă nu se utilizează cunoașterea euristică toate nodurile vor fi expandate numai o singură dată. Pentru ca algoritmul A* să nu expandeze același nod de două ori este nevoie să se impună următoarea condiție

$$h'(n_i) \leq h'(n_j) + c(n_i, n_j)$$

pentru orice nod n_i al grafului și succesorii săi n_j . Această condiție este cunoscută sub denumirea de restricția de monotonicitate, și este analizată prin următoarea teoremă.

Teorema 6.5. *Dacă restricția de monotonicitate este satisfăcută, la momentul cînd nodul n este expandat prin algoritmul A*, drumul optim ce trece prin n a fost găsit.*

Demonstrație. Reprezentînd drumul optim ce trece prin n ca succesiunea nodurilor $P = (n_0, n_1, \dots, n_k)$ (unde $n_0 = S$ și $n_k = n$), la momentul în care nodul n altul decît nodul de start este extras din `open_list` pentru a fi expandat se consideră că P nu a fost găsit. La acest moment, nodurile din `closed_list` sînt nodurile incluse în P (ultimul fiind S). Cu aceste noduri se presupune că n_i este ultimul în `open_list` și $n_{i+1} \neq n$. De la restricția de monotonicitate pentru un i arbitrar:

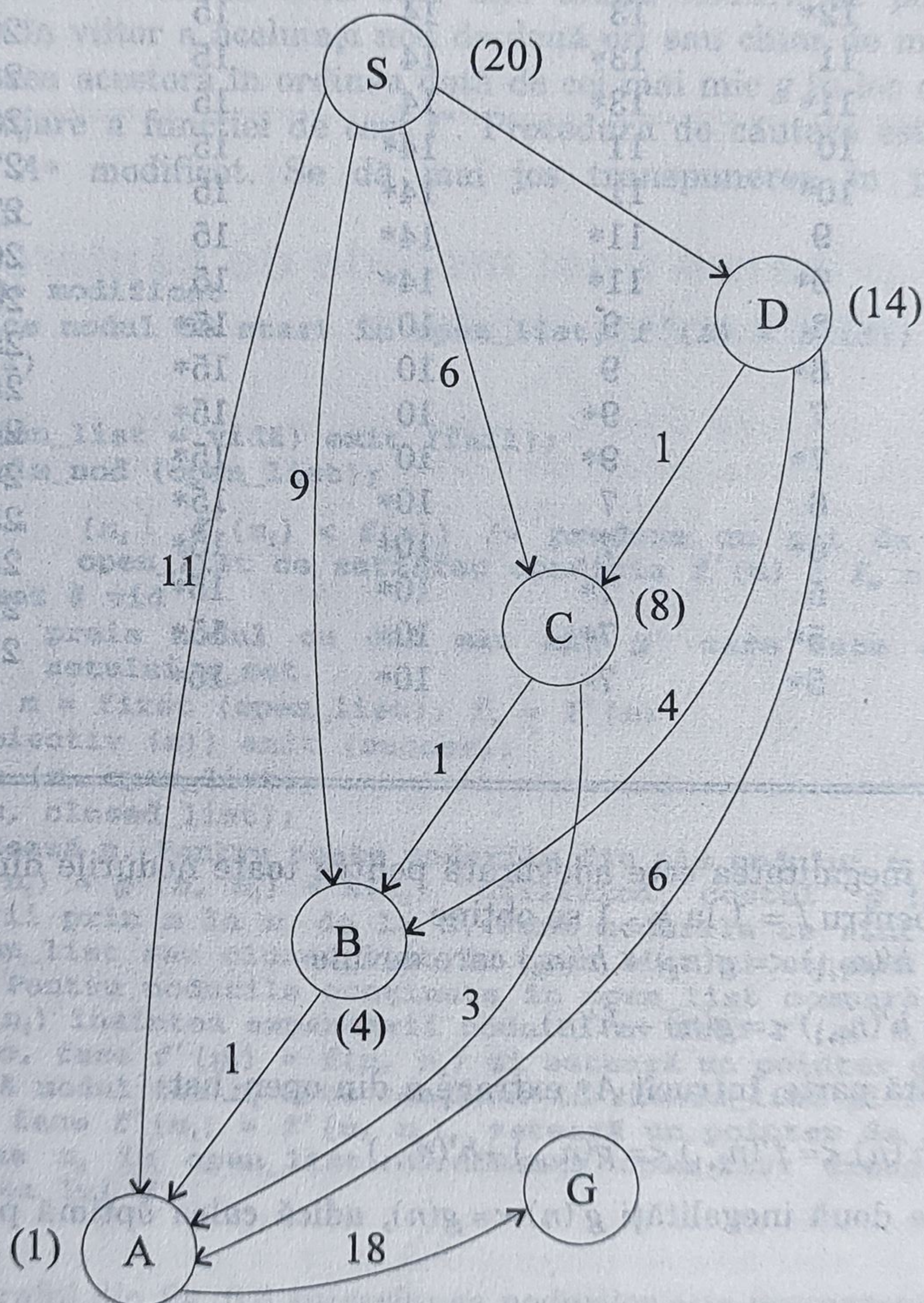
$$g(n_i) + h'(n_i) \leq g(n_i) + h'(n_{i+1}) + c(n_i, n_{i+1})$$

Întrucît n_i și n_{i+1} sînt conținute în calea optimă

$$g(n_{i+1}) = g(n_i) + c(n_i, n_{i+1}) \text{ și}$$

$$g(n_i) + h'(n_i) \leq g(n_{i+1}) + h'(n_{i+1}).$$

▼ Fig. 6.8. Graf în care un nod este expandat de mai multe ori prin algoritmul A*



▼ Tabela 6.1

S	A	B	C	D	G
20					
20*	12	13	14	15	29
20*	12*	13	14	15	29
20*	11	13*	14	15	28
20*	11*	13*	14	15	28
20*	10	11	14*	15	27
20*	10*	11	14*	15	27
20*	9	11*	14*	15	26
20*	9*	11*	14*	15	26
20*	8	9	10	15*	25
20*	8*	9	10	15*	25
20*	7	9*	10	15*	24
20*	7*	9*	10	15*	24
20*	6	7	10*	15*	23
20*	6*	7	10*	15*	23
20*	5	7*	10*	15*	22
20*	5*	7*	10*	15*	22*
20*	5*	7*	10*	15*	

Întrucît inegalitatea este adevărată pentru toate nodurile din P , dacă se utilizează pentru $l = 1$ la $k - 1$ se obține

$$g(n_{l+1}) + h'(n_{l+1}) \leq g(n_k) + h'(n_k) \text{ care devine}$$

$$g(n_{l+1}) + h'(n_{l+1}) \leq g(n) + h'(n).$$

Pe de altă parte, întrucît A^* extrage n din open_list

$$g'(n) + h'(n) \leq f'(n_{l+1}) \leq g(n_{l+1}) + h'(n_{l+1})$$

Din cele două inegalități $g'(n) \leq g(n)$, adică calea optimă prin n a fost găsită.

Întrucît graful din fig. 6.8 nu satisface restricția de monotonicitate, de mai multe ori un nod aflat pe drumul optim este expandat. Teorema 6.4 este aplicabilă dacă $h' = 0$ întrucît în acest caz restricția de monotocitate este satisfăcută. În plus rezultă că teorema 6.2 este doar un caz particular al teoremei 6.5.

O îmbunătățire a algoritmului prin care nodurile aflate pe drumul optim se expandează de mai puține ori va fi prezentată. Ea se bazează pe faptul că înainte de terminarea căutării, un nod n ce satisface $f'(n) \leq f(S)$ este prezent

în `open_list`, conform cu lema 6.2. Presupunînd că algoritmul A^* a expandat nodul n' , se obține $f'(n') \leq f'(n) \leq f(S)$. Dacă din nodurile expandate pînă la acest moment nodul avînd costul inferențiat maxim înaintea expandării este m , $f'(m) \leq f(S)$. Dacă $f'(n_i) < f'(m)$ pentru un nod n_i în `open_list` la acel moment, nodul n_i va fi inevitabil expandat. Rațiunea ce dictează aceasta arată că dacă $f'(n_i) < f'(m)$, se obține $f'(m) < f(S)$ și nodul scop $G(f(G) = f(S))$ nu poate fi extras din `open_list` atîta timp cît nodul n_i este încă aici. De aceea vor fi expandate toate nodurile ce satisfac restricția $f'(n_i) < f'(m)$ și numai după aceea nodul m . Cum în listă sînt mai multe noduri, se poate preveni expandarea în viitor a aceluiași nod de două ori sau chiar de mai multe ori prin aranjarea acestora în ordinea dată de cel mai mic g în loc de utilizarea pentru aranjare a funcției de cost f' . Procedura de căutare este numită și algoritmul A^* modificat. Se dă mai jos transpunerea în pseudocod a algoritmului.

Algoritm A^* modificat

Introduce nodul de start în `open_list`; $f'(S) = h'(S)$;

While (1)

{

if (`open_list` = vidă) exit (fail);

$n = \text{prim_nod}(\text{open_list})$;

$n_set = \{n_i \mid f'(n_i) < f'(m)\}$ /* produce un set de noduri în `open_list` ce satisfac condiția $f'(m) < f_m$ n_set */

if $n_set \neq \text{vidă}$

then preia nodul cu cel mai mic g' care este element al setului n_set

else $n = \text{first}(\text{open_list})$, $f_m = f'(n)$

if (obiectiv (n)) exit (succes);

remove (n , `open_list`);

add (n , `closed_list`);

expandează n . Pentru toate nodurile fiu ale nodului n calculează $f'(n, n_i) = g'(n, n_i) + h(n_i)$ utilizînd costul $g'(n, n_i)$ al trecerii prin n la n_i de la S . Pune nodurile ce sînt conținute în `open_list` sau `closed_list` în `open_list` și setează pointerii la n . Pentru nodurile conținute în `open_list` compară $f'(n_i)$ și $f'(n, n_i)$ înaintea expandării nodului n . Dacă $f'(n, n_i)$ este cel mai mic, face $f'(n_i) = f(n, n_i)$ și setează un pointer de la n_i la n . Dacă nodul fiu n_i este conținut în `closed_list` și $f'(n, n_i) < f'(n_i)$ face $f'(n_i) = f'(n, n_i)$, setează un pointer de la n la n_i și pune n_i în `open_list`. Ordonează `open_list` crescător după valoarea lui f' .

}

Pentru graful din fig. 6.8 succesiunea nodurilor este prezentată în tabelul 6.2. În tabel sînt prezentate nodurile din `open_list` și `closed_list` cînd căutarea se face cu algoritmul A^* modificat. De asemenea sînt prezentate și valorile funcțiilor de cost g' , h' , f_m . Pentru marcarea nodurilor din `closed_list` s-a utilizat simbolul *.

Se observă că se obține o reducere considerabilă a numărului de expandări față de aplicarea algoritmului A^* . Se poate demonstra că în acest caz numărul maxim de expandări este n^2 .

▼ Tabelul 6.2

S	A	B	C	D	G	f_m
0 + 20						0
0 + 20*	11 + 1	9 + 4	6 + 8	1 + 14		20
0 + 20	7 + 1	5 + 4	2 + 8	1 + 14*		20
0 + 20	5 + 1	3 + 4	2 + 8*	1 + 14*		20
0 + 20	4 + 1	3 + 4*	2 + 8*	1 + 14*	22 + 0	20
0 + 20	4 + 1*	3 + 4*	2 + 8*	1 + 14*	22 + 0*	20
0 + 20	4 + 1*	3 + 4*	2 + 8*	1 + 14*		22

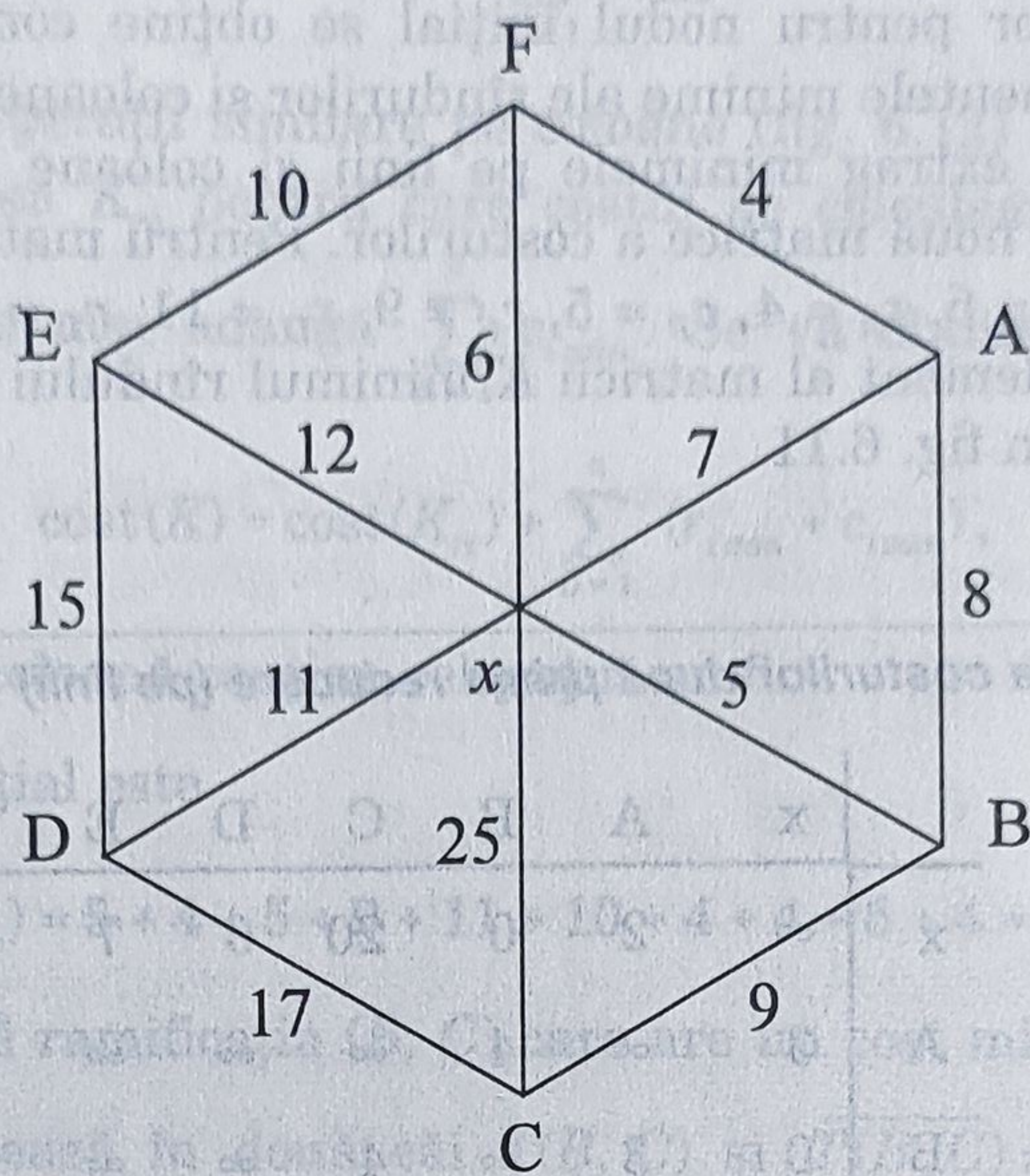
6.5. Aplicarea unei metode de căutare prin ramificare și limitare

Metodele prezentate pînă în prezent nu se bazează pe informația de selecție a alternativelor, decizia fiind luată pe baza corespondenței nodului cu obiectivele problemei, motiv pentru care se mai numesc și metode de cunoaștere informală. Îmbogățirea informației poate fi făcută prin atașarea la fiecare nod a unei informații referitoare la costul trecerii de la o stare (nod părinte) la alta (nod succesor sau fiu).

O problemă veche ce se încearcă a fi rezolvată prin metode de căutare cu ramificare și limitare este problema comis voiajorului. Formularea problemei arată că o persoană ce îndeplinește, funcția de comis voiajor și lucrează într-un oraș X trebuie să ajungă odată pe lună într-o serie de orașe în care are de dus obiecte sau are de luat anumite obiecte. Firma la care lucrează suportă costul transportului de la orașul X la fiecare din orașele de destinație. Problema comis voiajorului este de a stabili acel traseu pentru care pornind din orașul X să atingă toate orașele din traseu cu destinația finală orașul X , astfel încît costul transportului să fie minim. În afară de informațiile alternative ce reprezintă costul transportului între orașul X și orașele învecinate sînt necesare informații despre costul deplasării între orașele din traseu.

O primă componentă a metodei este cea referitoare la ramificare ce stabilește metodele de obținere a arborelui de căutare. Se poate considera ca informație de stare acea structură simbolică ce se pune în corespondență cu obiectivul problemei, structură ce ar fi dată de un circuit al orașelor. Informația elementară poate fi privită ca informație de stare, fiind dată de conceptul de localitate, legăturile între localități fiind corespondentul direct al drumului fizic dintre acestea. Se va nota cu $T(i, j)$ submulțimea circuitelor ce conține drumul elementar (i, j) ce se presupune a fi în drumul optim și cu $\overline{T(i, j)}$ submulțimea circuitelor ce nu conțin drumul elementar (i, j) . Astfel, dintre toate drumurile ce pornesc din nodul de start este probabil ca în soluția optimă să se găsească acela care are costul cel mai mic dintre toate drumurile elementare ale problemei.

▼ Fig. 6.9. Harta asociată orașelor



A doua componentă, cea de limitare se bazează pe faptul că fiecărui nod din spațiu îi este atașată o limită minimă a costului circuitului în mulțimea selectată de nodul respectiv. Dacă harta avînd costurile asociate este cea din fig. 6.9 se poate construi matricea costurilor (fig. 6.10). În strategia de reprezentare a matricii costurilor se pune pentru drumul între orașul A și A ca fiind ∞ întrucît odată ajuns aici acesta nu este un drum valid. La fel pentru orașele între care nu există legătură directă se asociază costului infinit.

▼ Fig. 6.10. Matricea costurilor

		x	A	B	C	D	E	F
K=	x	∞	7	5	25	11	12	6
	A	7	∞	8	∞	∞	∞	∞
	B	5	8	∞	9	∞	∞	∞
	C	25	∞	9	∞	17	∞	∞
	D	11	∞	∞	17	∞	15	∞
	E	12	∞	∞	∞	15	∞	10
	F	6	4	∞	∞	∞	10	∞

Circuitul elementar are selectat pe fiecare rînd și coloană cîte un drum elementar, soluția problemei fiind acea secvență de drumuri ce asigură trecerea prin toate orașele asigurînd în același timp cost minim. Limita minimă a costurilor pentru nodul inițial se obține considerînd că acesta conține numai elementele minime ale rîndurilor și coloanelor matricii de cost. Pentru aceasta se extrag minimele pe linii și coloane din fiecare rînd și coloană, obținînd o nouă matrice a costurilor. Pentru matricea de mai sus se obțin pe rînduri $r_x = 5$, $r_A = 4$, $r_B = 5$, $r_C = 9$, $r_D = 11$, $r_E = 10$, $r_F = 4$. Dacă se scade din fiecare element al matricii K minimul rîndului respectiv se obține matricea redusă din fig. 6.11.

▼ Fig. 6.11. Matricea costurilor după prima reducere (pe linii)

	x	A	B	C	D	E	F
x	∞	2	0	20	6	7	1
A	3	∞	4	∞	∞	∞	0
Kr = B	0	3	∞	4	∞	∞	∞
C	16	∞	0	∞	8	∞	∞
D	0	∞	∞	6	∞	4	∞
E	7	∞	∞	∞	5	∞	0
F	2	0	∞	∞	∞	6	∞

▼ Fig. 6.12. Matricea costurilor după a doua reducere (pe coloane)

	x	A	B	C	D	E	F
x	∞	2	0	16	1	3	1
A	3	∞	4	∞	∞	∞	0
Krc = B	0	3	∞	0	∞	∞	∞
C	16	∞	0	∞	3	∞	∞
D	0	∞	∞	2	∞	0	∞
E	7	∞	∞	∞	0	∞	0
F	2	0	∞	∞	∞	2	∞

Costul calculat pentru matricea K este echivalent cu cel calculat pentru matricea K_r , dacă la acesta se adaugă $\sum_{L=1}^n r_{i\min}$.

Aplicarea unei operații similare pe coloane (fig. 6.12) face să se obțină o nouă matrice redusă K_{rc} pentru care costul se calculează la fel cu cel al matricii K_r , însă se mai adaugă $\sum_{L=1}^n c_{i\min}$. Se va obține limita minimă a costului ca fiind:

$$\text{cost}(K) = \text{cost}(K_{rc}) + \sum_{L=1}^n (r_{i\min} + c_{i\min}),$$

iar fiecare linie și coloană conține cel puțin un 0.

Astfel costul inițial este

$C_0 = \sum_{L=1}^n (r_{i\min} + c_{i\min}) = 5 + 4 + 5 + 9 + 11 + 10 + 4 + 4 + 5 + 4 = 61$. Dacă din nodul inițial se selectează ramificația (B, C) care are un cost minim în matricea K drumul se expandează în două căi $T(B, C)$ și $\overline{T(B, C)}$. Calculul costului presupune eliminarea din matrice a rîndului B și a coloanei C , indicînd faptul că odată orașul atins nu se revine în el. De asemenea, se dă arcului (C, B) cost infinit în matrice, eliminînd astfel drumul de întoarcere. După eliminare se obține matricea din fig. 6.13 asupra căreia se aplică o nouă operație de reducere dacă după eliminarea liniei și coloanei au apărut linii sau coloane ce au minimul diferit de 0 (fig. 6.14).

▼ Fig. 6.13. Matricea după eliminarea drumului (B, C)

	x	A	B	D	E	F
x	∞	2	0	1	7	1
A	3	∞	4	∞	∞	0
C	16	∞	0	3	∞	∞
D	0	∞	∞	∞	0	∞
E	2	∞	∞	0	∞	0
F	2	0	∞	∞	2	∞

▼ Fig. 6.14. Noua matrice redusă

	x	A	B	D	E	F
x	∞	2	0	1	7	1
A	3	∞	4	∞	∞	0
C	13	∞	∞	0	∞	∞
D	0	∞	∞	∞	0	∞
E	2	∞	∞	0	∞	0
F	2	0	∞	∞	2	∞

Prin reducerea matricii din fig. 6.13 al cărei rezultat este 6.14 se adaugă la costul minim valoarea 3 care este minimul pe linia C diferit de zero. Se obține astfel $\text{cost}(T(B, C)) = 61 + 3 = 64$. Din acest moment se poate alege drumul elementar de cost minim (X, B) , și se elimină linia marcată cu X și coloana marcată cu B. La acest moment nu este necesară reducerea, întrucât matricea costurilor obținută conține pe fiecare linie și coloană cel puțin un zero. Se poate alege la acest moment traseul (F, A) de cost minim. La acest moment matricea costurilor se reduce după $r_A = 3$ obținând matricea din fig. 6.15. Costul asociat pînă în acest moment este $\text{cost}(T(B, C)) + 3 = 64 + 3 = 67$. La acest moment se poate alege drumul elementar (C, D) , în care după alegerea drumului (D, E) de cost minim, rămîne o matrice de ordin 2 în care există două drumuri minime (A, X) și (E, F) .

▼ Fig. 6.15. Matrice redusă după mai mulți pași

	x	D	E	F
A	0	∞	∞	∞
C	13	0	∞	∞
D	0	∞	0	∞
E	2	0	∞	0

Traseul de cost minim astfel obținut este X, B, C, D, E, F, A, X avînd valoarea costului egală cu 67. La acest moment poate fi construit algoritmul general al căutării prin ramificare și limitare corespunzător metodei de rezolvare a problemei comis voiajorului.

- Pas1:* se construiește matricea costurilor corespunzătoare datelor inițiale, stabilind pentru nodul inițial mulțimea soluțiilor posibile și limita costului minim $\text{cost}(T)$;
- Pas2:* se obține din matricea costurilor matricea redusă ce asignează costurile curente;
- Pas3:* se selectează un arc (i, j) pentru următoarea ramificație în arbore având drept criteriu de selecție costul minim;
- Pas4:* se partitionează mulțimea soluțiilor posibile după drumurile $T(i, j)$ și, $\overline{T(i, j)}$;
- Pas5:* se reduce matricea costurilor curente prin eliminarea rândului i și a coloanei j ce corespund arcului selectat (i, j) , se calculează limitele maxime și minime ale costului folosind relațiile:
- $$\text{cost}(T(i, j)) = \text{cost}(T \text{ anterior}) + \text{cost}(T(i, j))$$

$$\text{cost}(\overline{T(i, j)}) = \text{cost}(T \text{ anterior}) + \text{cost}(\overline{T(i, j)})$$

în care, $\text{cost}(T \text{ anterior})$ este limita minimă a costului nodului părinte din care s-a ramificat arcul (i, j) , iar $\text{cost}(T(i, j))$ și $\text{cost}(\overline{T(i, j)})$ sînt costurile suplimentare ale circuitului optim în matricea redusă la pasul anterior.

- Pas6:* dacă ultimul arc a fost selectat, adică matricea costurilor are dimensiunea 2×2 , s-a obținut drumul optim; altfel se reia procedura de la pasul 3.

Sînt situații în care matricea costurilor are termeni exprimați sub formă de variabile de stare, situație în care costul aplicării unui operator este determinat doar dacă se cunoaște contextul de stare în care se aplică. În acest caz limita minimă a costului nu mai este reprezentată de suma celei mai reduse selecții necorelate cu restricțiile problemei, ci se introduc corecțiile în selecția operatorilor de ramificare. În momentul în care costul traseului pe ramificația selectată inițial ca limită minimă a ajuns să depășească valori neoptimale pînă atunci se selectează o altă ramificație, obținînd condiții suficiente de parcurs o altă ramificație mai avantajoasă față de cea anterioară.

7

REZOLVAREA PROBLEMELOR PRIN DECOMPOZIȚIE

7.1. Descompunerea problemelor

În acest paragraf se descrie procesul de generare a unei propoziții, ca exemplu al modului în care o problemă se descompune. Procesul de descompunere în propoziții urmăriște restricțiile impuse de o gramatică dată. Se consideră că gramatica este dată prin următoarele reguli simple:

- R1 PROPOZIȚIE \rightarrow SUBIECT (SUB) PREDICAT (PRED)
- R2 SUB \rightarrow PRONUME (PRON)
- R3 SUB \rightarrow NUME PREDICATIV (NP)
- R4 PRED \rightarrow VERB (V) NUME PREDICATIV (NP)
- R5 NP \rightarrow ARTICOL NEHOTĂRÂT (DET) (N)
- R6 PRON \rightarrow He
- R7 V \rightarrow cat
- R8 DET \rightarrow a
- R9 N \rightarrow dog
- R10 N \rightarrow cat

În etajul superior cu litere mari sunt reprezentate simboluri necominate. În etajul inferior cu litere mici sunt reprezentate simboluri cominate. Formele de la stânga la dreapta se descompun în simboluri terminale prin aplicarea regulilor R1-R10. O secvență de simboluri se poate specifica pentru a fi o propoziție.

O strategie de rezolvare a problemelor pornește de la descompunerea acestora în probleme mai simple numite și subprobleme. Dacă problema poate fi descompusă în subprobleme, spațiul stărilor poate fi reprezentat prin grafuri ȘI/SAU, și pot fi aplicate metodele de căutare specifice pentru grafuri ȘI/SAU metode ce includ căutarea prin minimizarea costului, precum și metode de căutare euristică. Jocurile sînt probleme tipice reprezentate prin grafuri ȘI/SAU așa că o strategie de mutare într-un joc este de fapt redusă la o problemă de căutare în graf.

7.1. Descompunerea problemelor

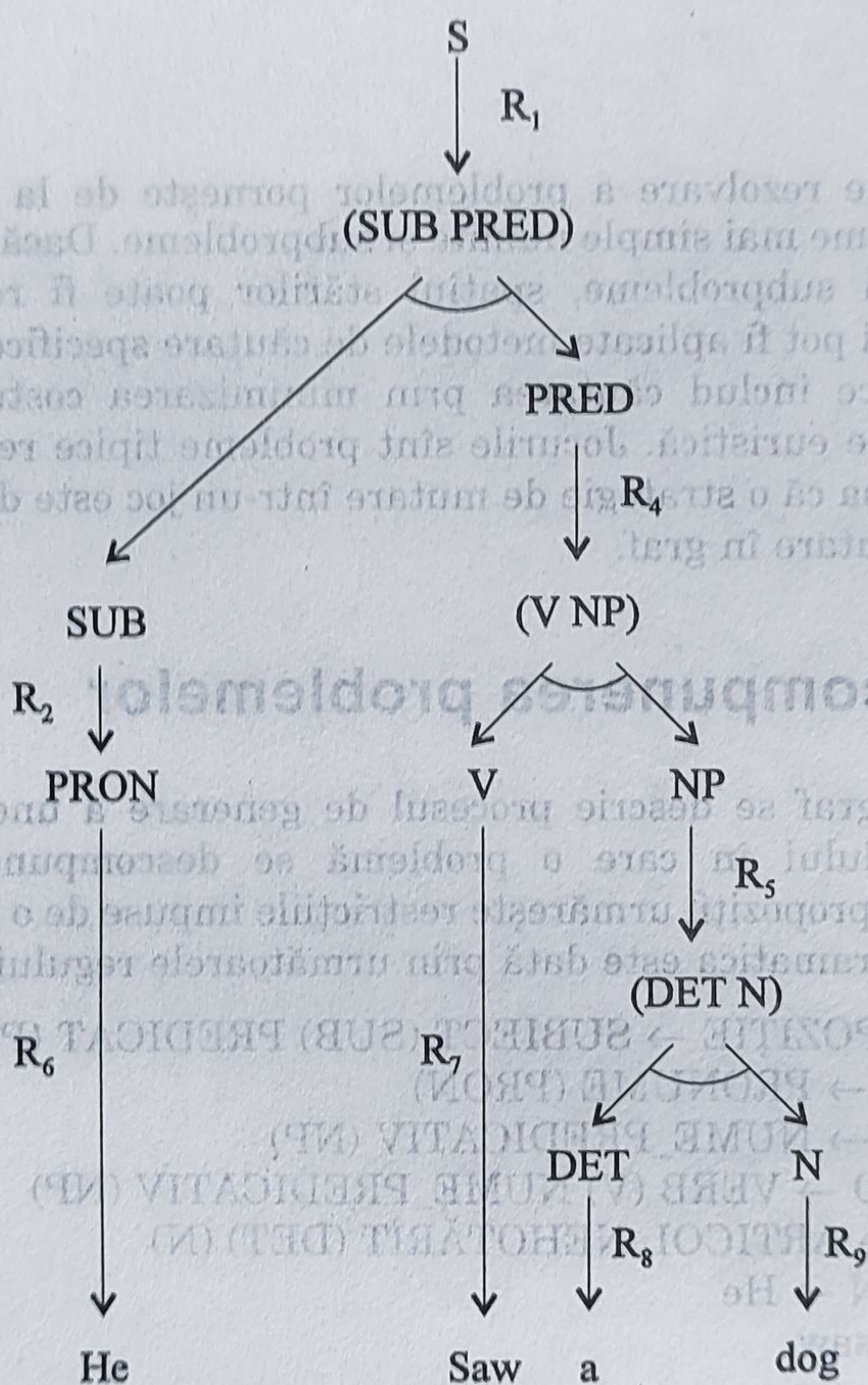
În acest paragraf se descrie procesul de generare a unei propoziții, ca exemplu al modului în care o problemă se descompune. Procesul de descompunere în propoziții urmărește restricțiile impuse de o gramatică dată. Se consideră că gramatica este dată prin următoarele reguli simple:

- R1 PROPOZIȚIE \rightarrow SUBIECT (SUB) PREDICAT (PRED)
- R2 SUB \rightarrow PRONUME (PRON)
- R3 SUB \rightarrow NUME_PREDICATIV (NP)
- R4 PRED \rightarrow VERB (V) NUME_PREDICATIV (NP)
- R5 NP \rightarrow ARTICOL NEHOTĂRÎT (DET) (N)
- R6 PRON \rightarrow He
- R7 V \rightarrow saw
- R8 DET \rightarrow a
- R9 N \rightarrow dog
- R10 N \rightarrow cat

În stînga scrise cu litere mari s-au reprezentat simboluri neterminale, în dreapta în regulile R6-R10 sînt reprezentate cu litere mici simboluri terminale. Pornind de la starea inițială S, se dorește construirea unei propoziții numai din simboluri terminale, prin aplicarea repetată a regulilor. O secvență de simboluri se poate specifica printr-o listă între paranteze.

Astfel prin aplicarea regulii R_1 , aplicabila în starea inițială S se obține (SUB PRED). Problema în acest moment este cea de conversie a listei într-o secvență de simboluri terminale. Subproblemele obținute prin descompunerea problemei, adică cele de conversie într-o listă de simboluri terminale a elementelor SUB și PRED, sînt mai simple și pot fi rezolvare independent și aplicate în paralel. Prin aplicarea regulii R_4 la PRED se obține lista (V NP), deci o descompunere în subproblemele V și NP. Figura 7.1 arată un exemplu al procedurii pentru obținerea unei soluții.

▼ Fig. 7.1. Exemplu de generare a unei propoziții



Pentru a indica faptul că problema a fost descompusă în figură s-au specificat legături între drumurile grafului. Dacă se notează timpul pentru aplicarea unei reguli $t(R)$ și diferite reguli pot fi aplicate independent, se poate presupune că sînt aplicate în paralel. Notînd cu $f(n)$ costul atingerii simbolului terminal de la nodul n se obține:

$$f(\text{PRON}) = t(R_6), f(V) = t(R_7), f(\text{DET}) = t(R_8), f(N) = t(R_9)$$

Dacă se consideră că n este rezolvat prin nodurile fiu n_1, n_2, \dots, n_m se obține:

$$f(n) = \max\{f(n_1), f(n_2), \dots, f(n_m)\}$$

Ca urmare, $f(NP) = t(R5) + \max\{f(DET), f(N)\}$. Făcând aceste calcule pentru toate nodurile se obține costul total $f(S)$. Dacă regulile nu sînt aplicate în paralel se obține pentru costul nodului n

$$f(n) = f(n_1) + f(n_2) + \dots + f(n_m) = \sum_{i=1}^m f(n_i).$$

Ultimele două relații sînt exemple tipice pentru calculul costului atunci cînd o problemă este descompusă. Una sau alta din aceste formule este practic utilizată la orice problemă. În calcul, nu s-a luat în considerare costul descompunerii, în cazul de față fiind considerat inclus în costul arcului dinaintea nodului corespunzînd descompunerii. Ca exemplu costul descompunerii (SUB PRED) este inclus în costul aplicării regulii $R1$.

7.2. Grafuri ȘI/SAU

Problema generării propoziției poate fi reprezentată prin grafuri ȘI/SAU ca în fig. 7.2. În acest caz graful este un arbore. La problemele reprezentate în spațiul stărilor este suficient ca un singur nod soluție să fie atins, pe cînd la grafurile ȘI/SAU procesul descompunerii desface problema în subprobleme.

Astfel în fig. 7.2 nodurile SUB și PRED formează o structură de nod ȘI. Pentru rezolvarea problemei corespunzătoare nodului părinte al nodurilor ȘI, trebuiesc rezolvate toate subproblemele corespunzînd nodurilor fiu ale acestuia. În figură nodurile ȘI au fost marcate prin arce, nodurile nemarcate fiind noduri SAU. Nodurile SAU sînt identice cu cele prezentate în capitolul 6 referitoare la tehnici de căutare în arbori. Soluția problemei cu structura din fig. 7.1 este o soluție a grafului ȘI/SAU. Un graf parțial este numit *graf rezolvent*, iar rezolvarea unei probleme reprezentată prin grafuri ȘI/SAU invocă găsirea tuturor grafurilor rezolvente ale acesteia. În capitolul 6 au fost enumerate principalele caracteristici ale grafurilor ȘI/SAU. Graful din fig. 7.2 este un arbore ȘI/SAU. Dacă două noduri (cum este cazul nodului NP) sînt reprezentate ca un singur nod se obține un graf ȘI/SAU, nu un arbore ȘI/SAU. Aici, intenția de a reprezenta un nod de două ori este justificată de dorința de a obține mai multe soluții. În cazul de față soluția "a dog saw a cat" a fost obținută. În fig. 7.3a este prezentată o problemă complexă A ce poate fi rezolvată prin descompunerea în subproblemele B și C sau prin rezolvarea lui D. În graful respectiv se regăsesc ambele tipuri de noduri pentru același părinte. În fig. 7.3b nodurile au fost separate prin introducerea unui nod suplimentar pentru care funcția de cost este egală cu cea de descompunere a problemei A în subproblema M formată din B și C.

Dacă se consideră că n este rezolvat prin nodurile fiu n_1, n_2, \dots, n_m se obține:

$$f(n) = \max\{f(n_1), f(n_2), \dots, f(n_m)\}$$

Ca urmare, $f(NP) = t(R5) + \max\{f(DET), f(N)\}$. Făcând aceste calcule pentru toate nodurile se obține costul total $f(S)$. Dacă regulile nu sînt aplicate în paralel se obține pentru costul nodului n

$$f(n) = f(n_1) + f(n_2) + \dots + f(n_m) = \sum_{i=1}^m f(n_i).$$

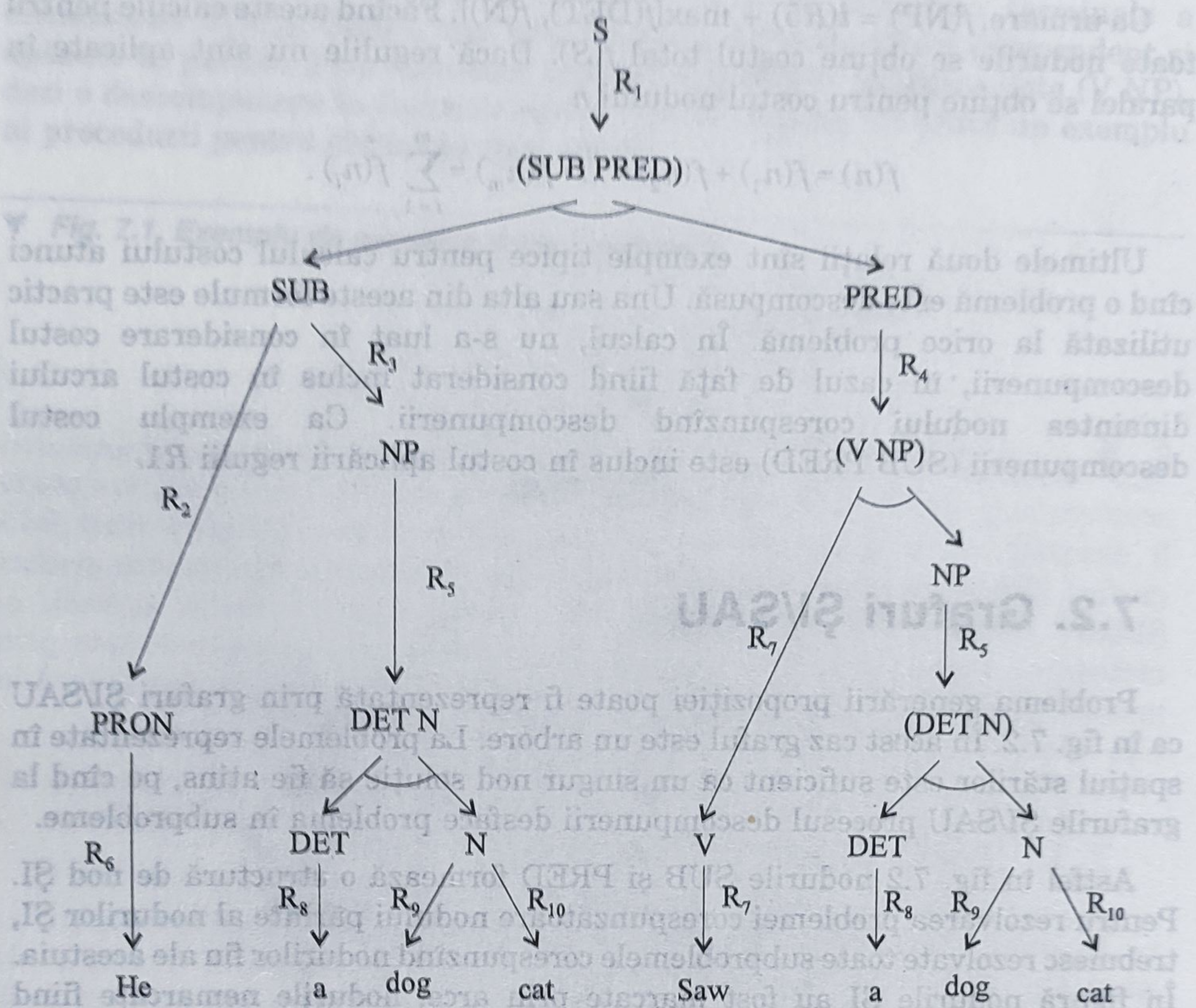
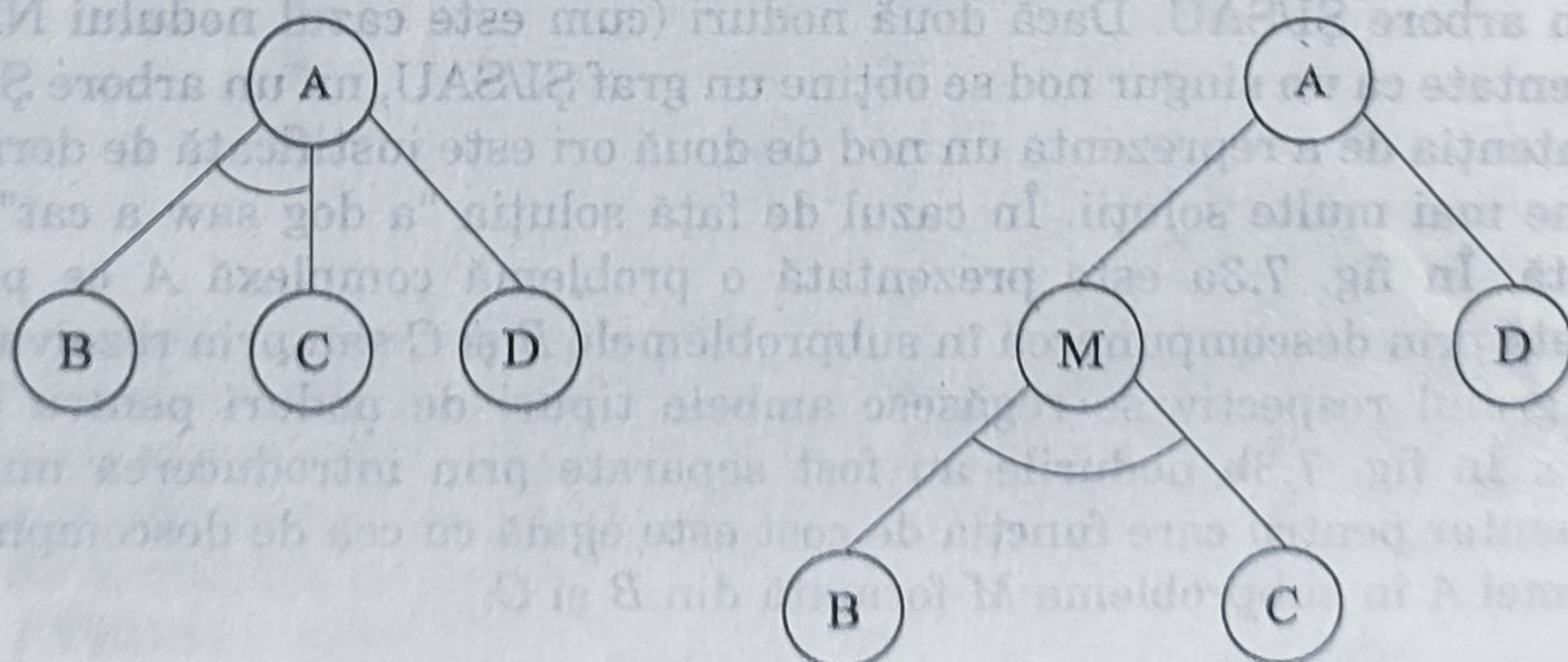
Ultimele două relații sînt exemple tipice pentru calculul costului atunci cînd o problemă este descompusă. Una sau alta din aceste formule este practic utilizată la orice problemă. În calcul, nu s-a luat în considerare costul descompunerii, în cazul de față fiind considerat inclus în costul arcului dinaintea nodului corespunzînd descompunerii. Ca exemplu costul descompunerii (SUB PRED) este inclus în costul aplicării regulii $R1$.

7.2. Grafuri ȘI/SAU

Problema generării propoziției poate fi reprezentată prin grafuri ȘI/SAU ca în fig. 7.2. În acest caz graful este un arbore. La problemele reprezentate în spațiul stărilor este suficient ca un singur nod soluție să fie atins, pe cînd la grafurile ȘI/SAU procesul descompunerii desface problema în subprobleme.

Astfel în fig. 7.2 nodurile SUB și PRED formează o structură de nod ȘI. Pentru rezolvarea problemei corespunzătoare nodului părinte al nodurilor ȘI, trebuie rezolvate toate subproblemele corespunzînd nodurilor fiu ale acestuia. În figură nodurile ȘI au fost marcate prin arce, nodurile nemarcate fiind noduri SAU. Nodurile SAU sînt identice cu cele prezentate în capitolul 6 referitoare la tehnici de căutare în arbori. Soluția problemei cu structura din fig. 7.1 este o soluție a grafului ȘI/SAU. Un graf parțial este numit *graf rezolvent*, iar rezolvarea unei probleme reprezentată prin grafuri ȘI/SAU invocă găsirea tuturor grafurilor rezolvente ale acesteia. În capitolul 6 au fost enumerate principalele caracteristici ale grafurilor ȘI/SAU. Graful din fig. 7.2 este un arbore ȘI/SAU. Dacă două noduri (cum este cazul nodului NP) sînt reprezentate ca un singur nod se obține un graf ȘI/SAU, nu un arbore ȘI/SAU. Aici, intenția de a reprezenta un nod de două ori este justificată de dorința de a obține mai multe soluții. În cazul de față soluția "a dog saw a cat" a fost obținută. În fig. 7.3a este prezentată o problemă complexă A ce poate fi rezolvată prin descompunerea în subproblemele B și C sau prin rezolvarea lui D . În graful respectiv se regăsesc ambele tipuri de noduri pentru același părinte. În fig. 7.3b nodurile au fost separate prin introducerea unui nod suplimentar pentru care funcția de cost este egală cu cea de descompunere a problemei A în subproblema M formată din B și C .

▼ Fig. 7.2. Graf ȘI/SAU al problemei de generare a propoziției

▼ Fig. 7.3. Metode de reprezentare a nodurilor ȘI/SAU mixate
a) noduri ȘI, SAU sînt fii ale aceluiași nod
b) graf ȘI/SAU

Un nod într-un arbore ȘI/SAU poate fi un nod ȘI al unui nod părinte dat și în același timp un nod SAU al altui nod părinte. Ca urmare, este necesar să se identifice relația părinte fiu. În acest material se vor utiliza expresiile " n_i este un nod ȘI al nodului n " sau "nodul n_i al lui n este un nod ȘI".

7.3. Căutarea în grafuri ȘI/SAU

Se tratează în continuare deferite metode pentru găsirea soluției într-un graf ȘI/SAU. Când se caută într-un graf este suficient să se găsească un singur drum, așa că dacă se cunosc pointerii de la nodurile părinte către toți fii săi în drumul parcurs, și căutarea este completă, înseamnă că s-a găsit o cale ce constituie o soluție. Totuși un graf rezolvent al grafului ȘI/SAU este la rîndul său un graf. Procesul găsirii soluției prin căutare corespunde dezvoltării progresive a soluției. Numai unele din grafurile rezolvente posibile sînt găsite în timpul căutării. Acestea sînt numite *grafuri candidate*. S-a preferat această denumire în loc de *grafuri rezolvente parțiale* întrucît nu se cunoaște încă dacă acestea sînt parte a soluției. Uzual aceste grafuri sînt găsite în timpul căutării și structurile lor trebuiesc reprezentate.

7.3.1. Evaluarea și expandarea grafurilor candidate

Un graf ȘI/SAU poate fi cercetat prin evaluări și expandări repetate după cum urmează:

1. Determină dacă nodurile grafurilor candidate sînt solvabile, insolvabile sau nedecidabile, precum și costul acestora dacă este cazul.
2. Expandează grafurile candidate.

În primul pas examinează toate punctele finale (nodurile frunză) ale grafului candidat. Dacă acestea sînt noduri terminale ele corespund unei probleme rezolvate și ca urmare vor fi marcate în acest sens. Dacă punctele terminale ale grafului candidat sînt puncte finale ale problemei inițiale, dar nu noduri terminale, problemele corespunzătoare acestor noduri sînt insolvabile și se vor marca printr-un marker UNSOLVED. Dacă nodurile fiu ale unui nod, altele decît nodurile terminale ale grafului candidat sînt noduri ȘI, și dacă toți fii sînt SOLVED, atunci nodul va fi marcat SOLVED. Dacă cel puțin un fiu este UNSOLVED, atunci nodul părinte va fi marcat ca UNSOLVED. În cazul în care nodurile fiu, altele decît punctele terminale sînt noduri SAU și cel puțin unul a fost marcat solved atunci nodul părinte va purta aceeași marcă.

Cînd nodul S al unui graf candidat devine SOLVED, graful este o soluție a problemei. Dacă este necesar calculul costului, se calculează costul tuturor nodurilor utilizînd costul punctelor finale ale grafurilor candidate. Costul nodului părinte al unui nod ȘI se calculează ca suma costurilor tuturor nodurilor sale fiu, pe cînd cel al unui nod SAU ca maximum costurilor fiilor. Costul nodului de start este costul grafului candidat. În pasul 2 se găsesc nodurile fiu prin expandarea nodurilor ce sînt puncte terminale ale grafului

candidat p și nu au fost încă marcate în nici un fel. Dacă nodurile fiu ale nodului expandat n sînt noduri ȘI un nou graf candidat este generat prin adăugarea tuturor nodurilor fiu la nodul p . Dacă nodurile fiu ale nodului n sînt noduri SAU adăugarea cîte unuia din acestea la p va determina generarea mai multor grafuri cîte noduri fiu are. În acest mod un mare număr de grafuri candidate vor fi generate prin procesul de căutare.

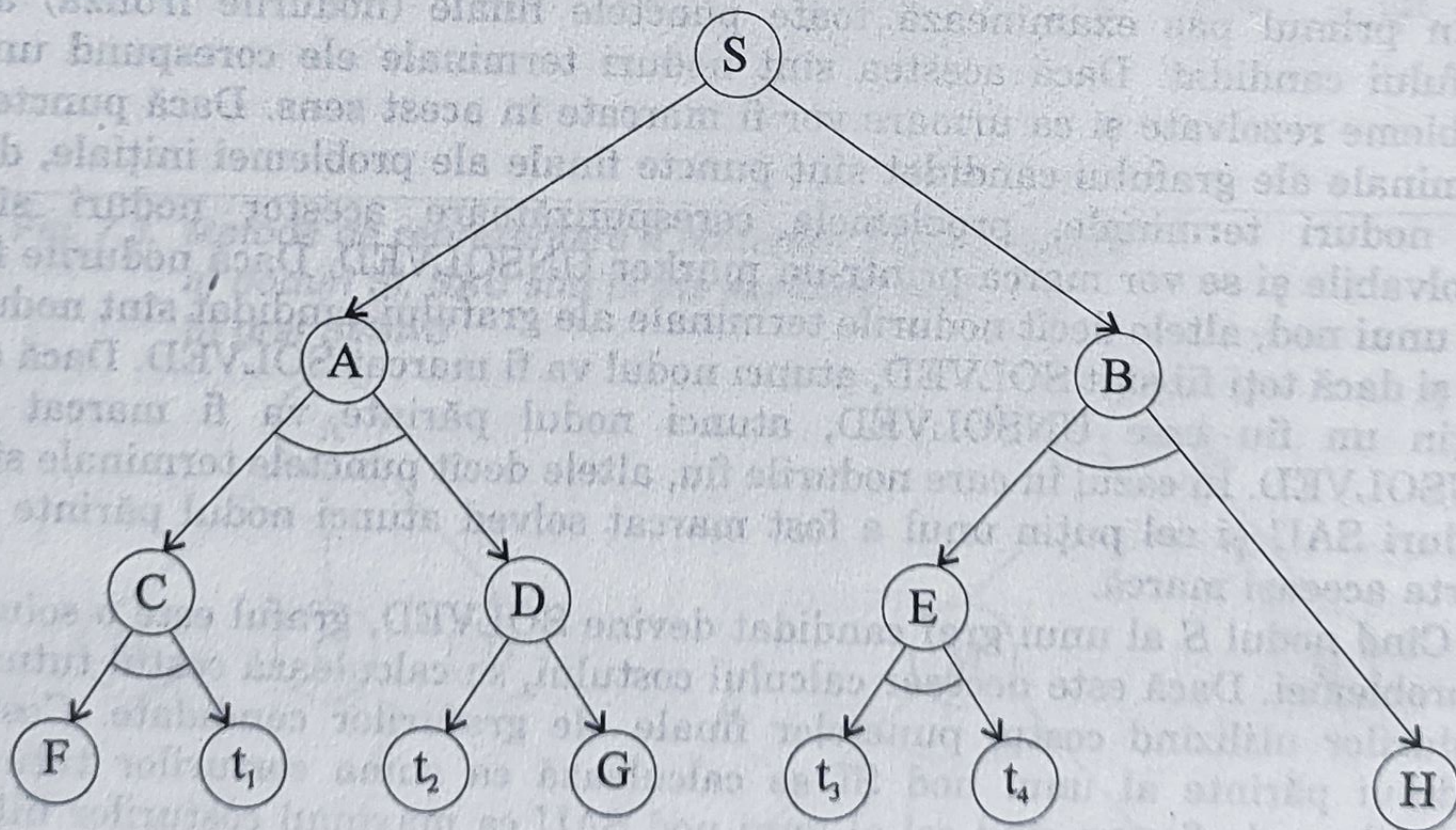
7.3.2. Căutarea în adîncime în grafuri ȘI/SAU

Ca și în cazul căutării în grafuri obișnuite, strategiile pentru căutarea în grafurile ȘI/SAU sînt: căutare în adîncime, căutare în lărgime, strategii de căutare optimă. Procedura de căutare în adîncime este prezentată în pseudocod mai jos.

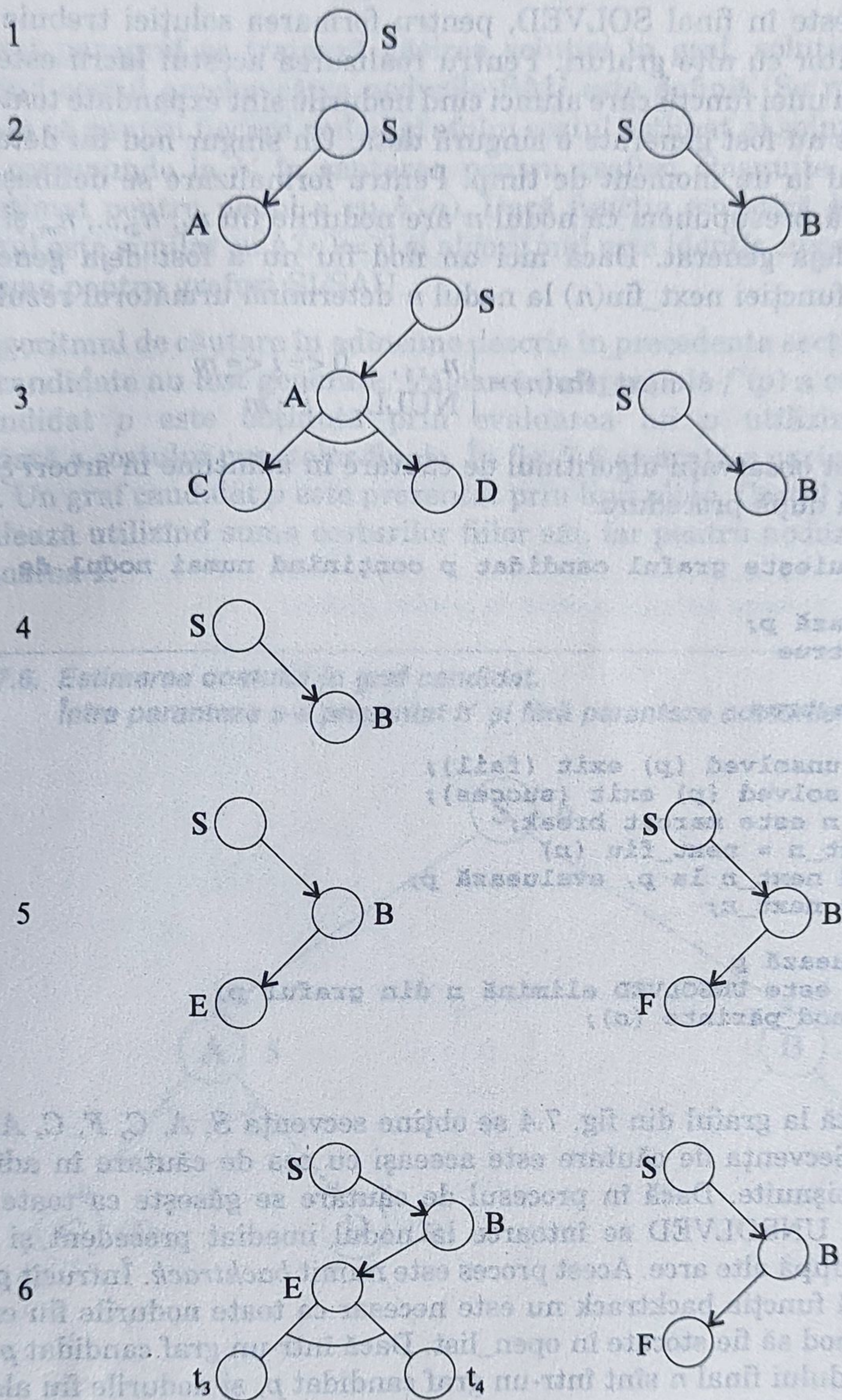
```

Introduce în open_list un graf candidat format doar din nodul de
start
While (1)
{
  if (open_list = vidă) exit (fail);
  p = prim_graf (open_list); /* aduce primul graf din listă
  if (solved (p)) exit (succes); /* dacă p este un graf rezolvent
    căutarea se termină și p este soluția */
  remove (p, open_list)
  add (n, closed_list);
  expandează p, evaluează toate noile grafuri candidate și pune
  grafurile candidate în care nodul de start nu este UNSOLVED în
  open_list;
}
  
```

▼ Fig. 7.4. Căutarea în grafuri ȘI/SAU, t_1 - t_4 sînt noduri terminale



▼ Fig. 7.5. Grafuri în open_list la căutarea în adîncime



În fig. 7.5 se arată grafurile candidate în open_list pentru graful ȘI/SAU din figura 7.4. Când primul graf candidat în open_list este expandat și evaluat la a treia parcurgere a buclei este găsit UNSOLVED și este mutat din open_list. La a șasea repetare a procedurii graful candidat din open_list este SOLVED și devine soluție a problemei.

În această procedură grafurile candidate expandate sînt ținute în *open_list*, însă nu este esențial a face acest lucru în cazul căutării în adîncime. Se poate utiliza un sistem în care această listă conține numai un graf parțial și dacă starea sa este în final SOLVED, pentru formarea soluției trebuie combinat corespunzător cu alte grafuri. Pentru realizarea acestui lucru este necesară construirea unei funcții care atunci cînd nodurile sînt expandate toate nodurile fiu ale sale au fost generate o singură dată. Un singur nod fiu determină un graf parțial la un moment de timp. Pentru formalizare se definește funcția *next_fiu*. Să presupunem că nodul n are nodurile fiu n_1, n_2, \dots, n_m și nodul fiu n_i a fost deja generat. Dacă nici un nod fiu nu a fost deja generat $i = 0$. Aplicarea funcției *next_fiu*(n) la nodul n determină următorul rezultat

$$\text{next_fiu}(n) = \begin{cases} n_{i+1}, & 0 \leq i \leq m \\ \text{NULL}, & i = m \end{cases}$$

Cu acest observații algoritmul de căutare în adîncime în arbori ȘI/SAU se desfășoară după procedura:

```

Construiește graful candidat  $p$  conținînd numai nodul de start
 $n = s$ ;
Evaluează  $p$ ;
while true
{
    while true
    {
        if unsolved ( $p$ ) exit (fail);
        if solved ( $p$ ) exit (succes);
        if  $n$  este marcat break;
        next_n = next_fiu ( $n$ )
        add next_n la  $p$ , evaluează  $p$ ;
         $n = \text{next\_n}$ ;
    }
    evaluează  $p$ 
    if  $n$  este UNSOLVED elimină  $n$  din graful  $p$ ;
     $n = \text{nod\_părinte} (n)$ ;
}

```

Aplicată la graful din fig. 7.4 se obține secvența $S, A, C, F, C, A, S, B, E, t_3, E, t_4$. Secvența de căutare este aceeași cu cea de căutare în adîncime în grafuri obișnuite. Dacă în procesul de căutare se găsește ca toate nodurile SAU sînt UNSOLVED se întoarce la nodul imediat precedent și continuă căutarea după alte arce. Acest proces este numit *backtrack*. Întrucît procedura realizează funcția *backtrack* nu este necesar ca toate nodurile fiu expandate ale unui nod să fie stocate în *open_list*. Dacă într-un graf candidat p nodurile fiu ale nodului final n sînt într-un graf candidat p , și nodurile fiu ale nodului final n sînt noduri ȘI se adaugă toate la graful p . Pentru a reduce timpul cînd unul din nodurile fiu este UNSOLVED n devine UNSOLVED și nu necesită găsirea altor noduri fiu. Pentru a continua procesul se utilizează funcția *next_fiu*.

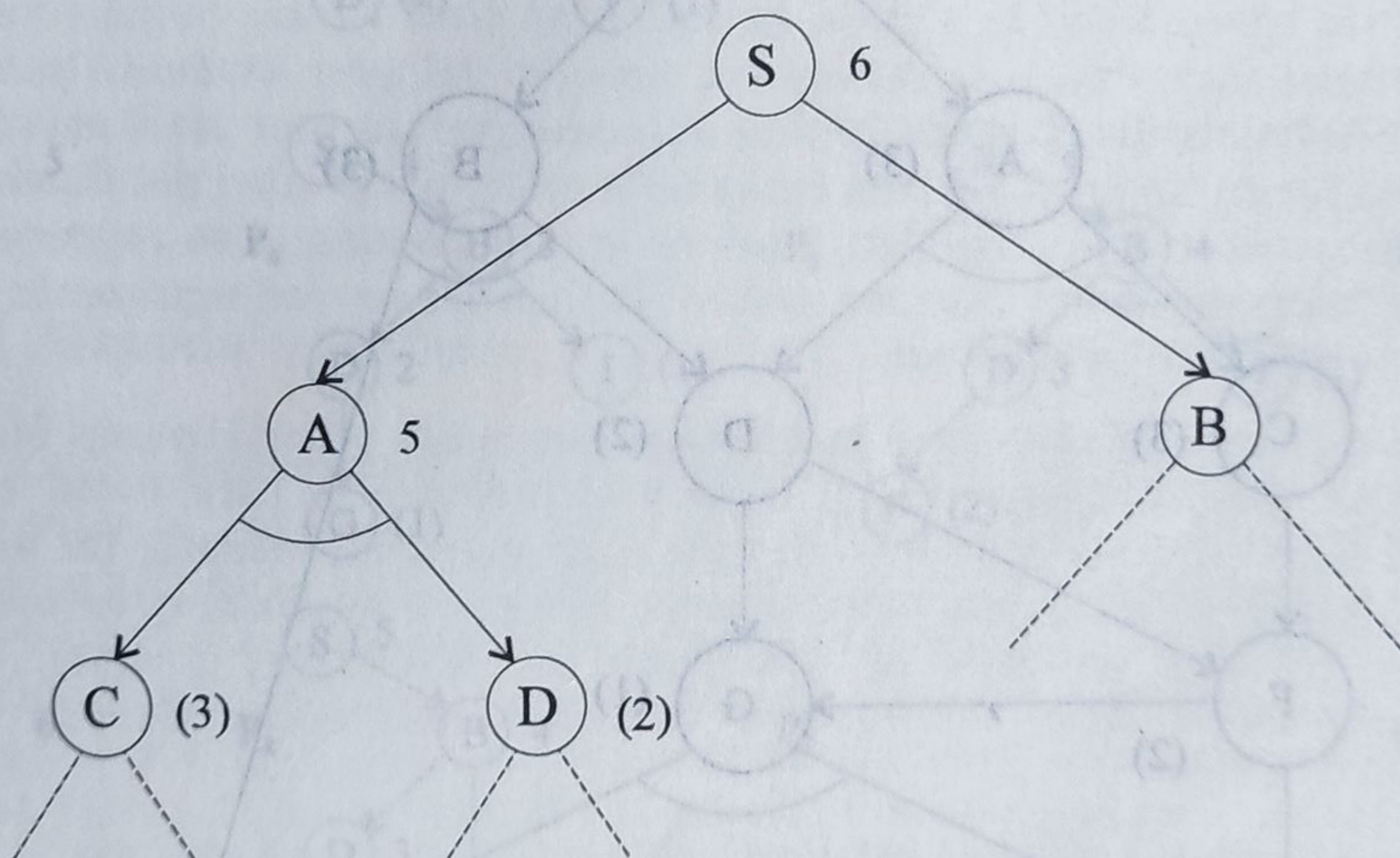
7.3.3. Căutarea soluției optime în grafuri ȘI/SAU

În acest paragraf se tratează găsirea soluției în graf, soluție avînd cost minim cînd costul arcelor către nodurile SAU este definit. Se presupune în continuare că pentru fiecare nod al grafului costul estimat al soluției este dat. Aceasta corespunde la h' în căutarea pentru grafuri obișnuite. Se va nota costul estimat pentru nodul n cu $h'(n)$. Dacă funcția euristică $h'(n)$ nu este dată, cazul este similar cu $h'(n) = 0$ și algoritmul este identic cu cel de căutare în adîncime pentru grafuri ȘI/SAU.

La algoritmul de căutare în adîncime descris în precedenta secțiune diferite grafuri candidate au fost generate. Valoarea inferențială $f'(p)$ a costului unui graf candidat p este obținută prin evaluarea lui p utilizînd valoarea inferențiată a costului punctelor finale. În fig. 7.6 se arată o parte a unui graf ȘI/SAU. Un graf candidat p este prezentat prin linii pline. Costul nodurilor ȘI se calculează utilizînd suma costurilor fiilor săi, iar pentru nodurile SAU s-a luat valoarea 1.

▼ Fig. 7.6. Estimarea costului în graf candidat.

Între paranteze s-a prezentat h' și fără paranteze costurile evaluate



Pune în open_list graful candidat p format numai din nodul de start S

$$f'(p) = h'(S);$$

while true

{

if (open_list = vidă) exit (fail);

$p = \text{prim_graf}(\text{open_list});$ /* aduce primul graf din listă

if (solved (p)) exit (succes); /* dacă p este un graf rezolvent

căutarea se termină și p este soluția */

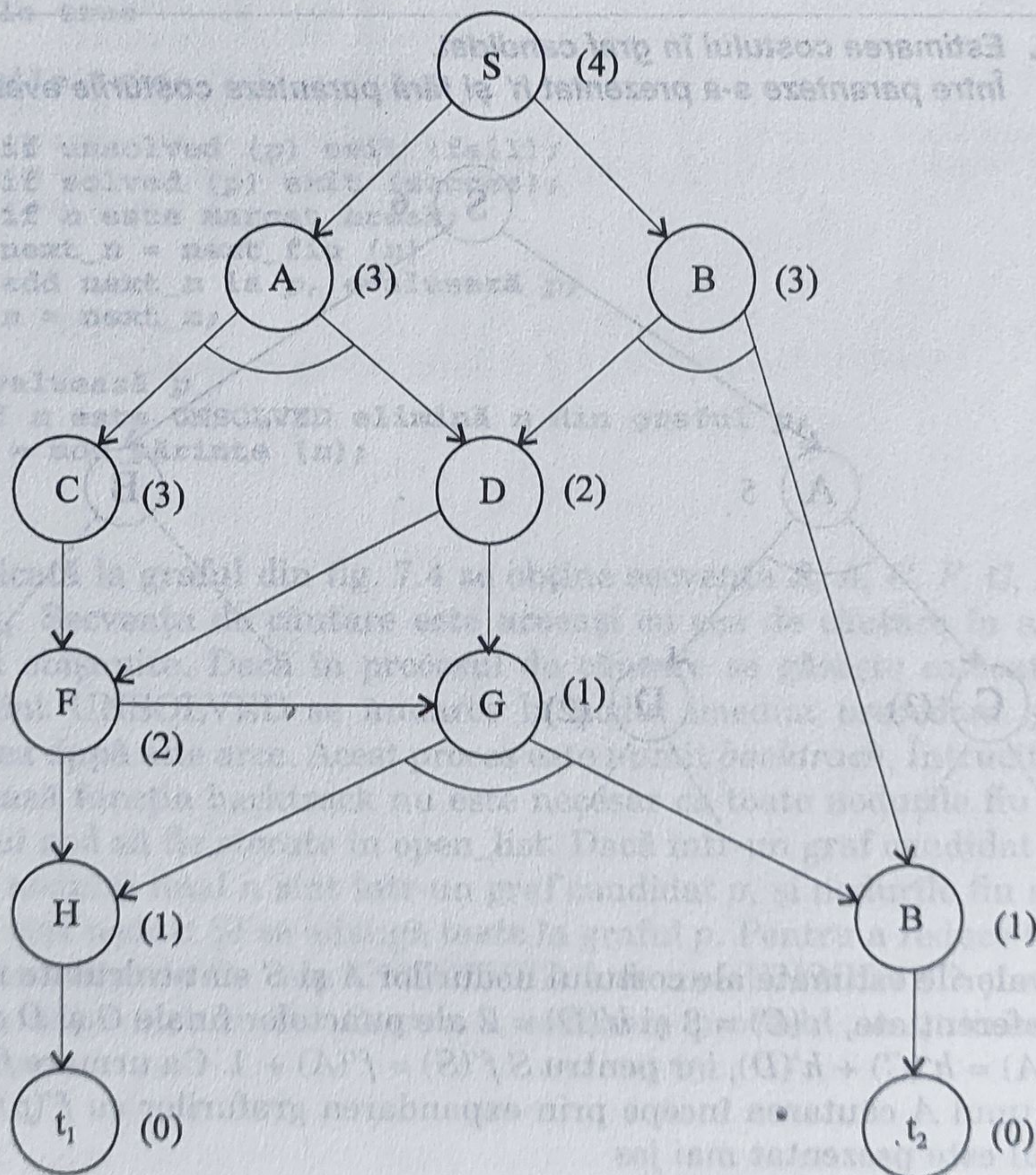
remove (p , open_list).

expandează p , evaluează toate grafurile candidate p_i nou generate și găsește $f'(p_i)$. Dacă p_i este UNSOLVED îl pune în open_list.

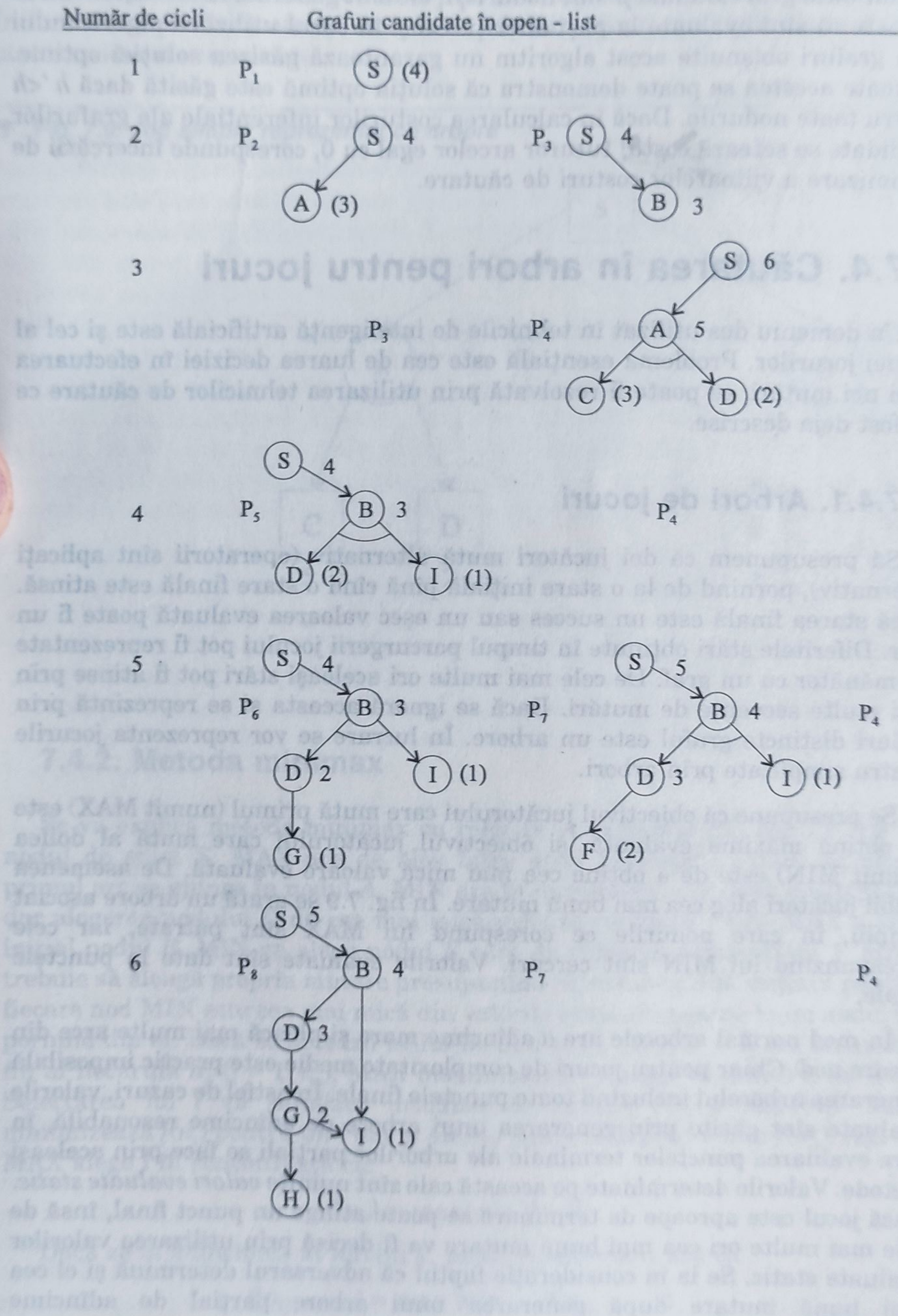
}

Pentru a ilustra modul în care algoritmul avansează se consideră exemplul din fig. 7.7, iar în fig. 7.8 se prezintă conținutul în open_list prin utilizarea algoritmului de căutare optimală. În figuri s-au trecut între paranteze costurile estimate, costurile fără paranteze sînt calculate utilizînd costuri estimate. Fiecare graf candidat este denumit, pentru a nu desena de mai multe ori același graf.

▼ Fig. 7.7. Căutarea soluției optime în grafuri ȘI/SAU



▼ Fig. 7.8. Soluția căutării optimale pentru graful din fig. 7.7



În fig. 7.8 sînt arătați numai primii șase ciclii. După ultimul ciclu ilustrat, H și I sînt expandate, și nodurile terminale t_1 și t_2 sînt adăugate obținînd graful p_{10} . Costul pentru p_{10} este $f(p_{10}) = 5$. Dacă nodurile fiu ale unui nod n asociat unui graf candidat p sînt noduri ȘI, ele sînt generate la același moment cu toate că sînt evaluate la pași diferiți. Ca și în cazul utilizării algoritmului A în grafuri obișnuite acest algoritm nu garantează găsirea soluției optime. Cu toate acestea se poate demonstra că soluția optimă este găsită dacă $h' < h$ pentru toate nodurile. Dacă în calcularea costurilor inferentiale ale grafurilor candidate se setează costul tuturor arcelor egal cu 0, corespunde încercării de minimizare a viitoarelor costuri de căutare.

7.4. Căutarea în arbori pentru jocuri

Un domeniu des utilizat în tehnicile de inteligență artificială este și cel al teoriei jocurilor. Problema esențială este cea de luarea deciziei în efectuarea unei noi mutări, ce poate fi rezolvată prin utilizarea tehnicilor de căutare ce au fost deja descrise.

7.4.1. Arbori de jocuri

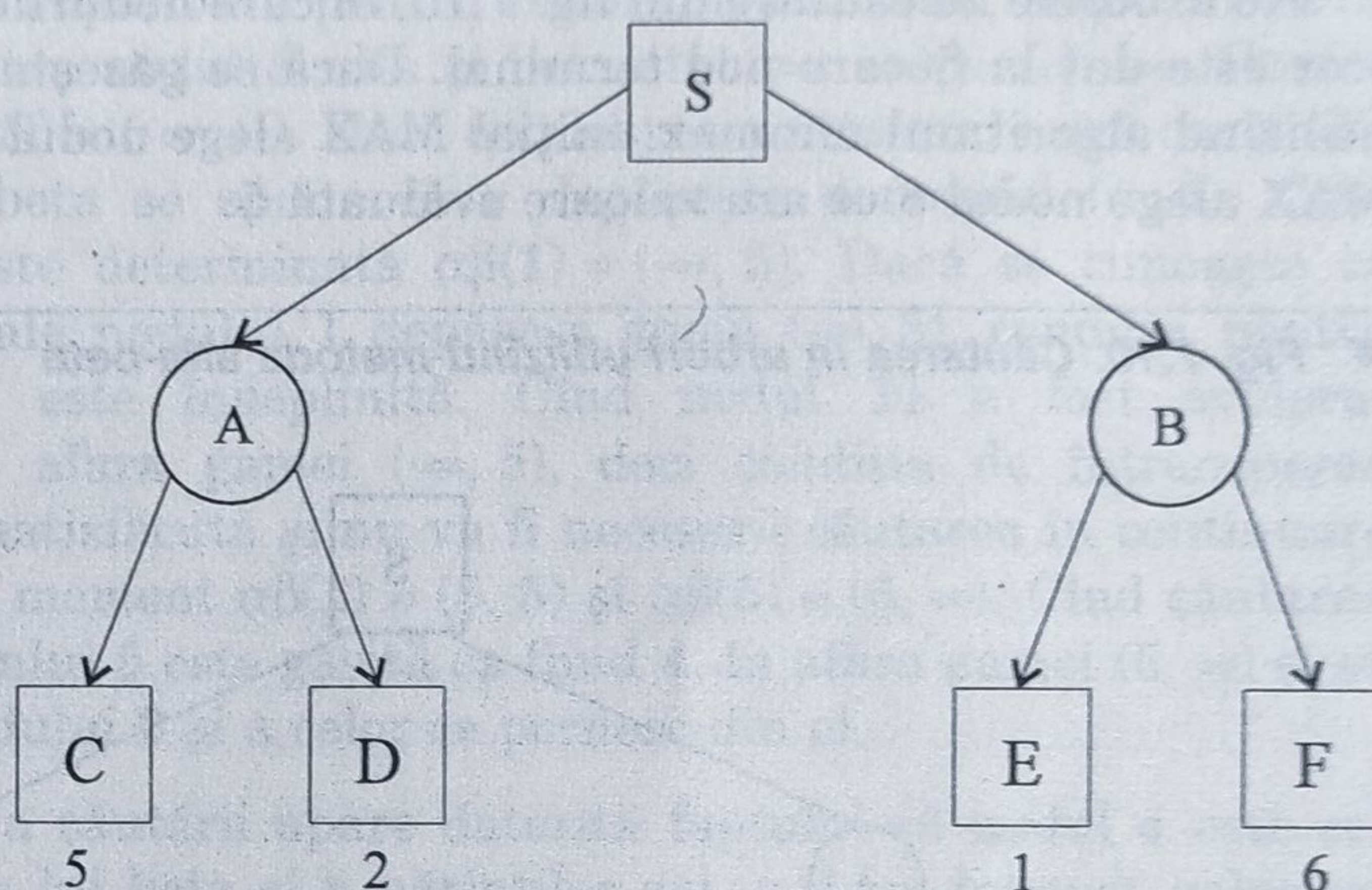
Să presupunem că doi jucători mută alternativ (operatorii sînt aplicați alternativ), pornind de la o stare inițială pînă cînd o stare finală este atinsă. Dacă starea finală este un succes sau un eșec valoarea evaluată poate fi un scor. Diferitele stări obținute în timpul parcurgerii jocului pot fi reprezentate asemănător cu un graf. De cele mai multe ori aceleași stări pot fi atinse prin mai multe secvențe de mutări. Dacă se ignoră aceasta și se reprezintă prin noduri distincte graful este un arbore. În lucrare se vor reprezenta jocurile pentru simplitate prin arbori.

Se presupune că obiectivul jucătorului care mută primul (numit MAX) este să obțină maxima evaluată, și obiectivul jucătorului care mută al doilea (numit MIN) este de a obține cea mai mică valoare evaluată. De asemenea ambii jucători aleg cea mai bună mutare. În fig. 7.9 se arată un arbore asociat simplu, în care nodurile ce corespund lui MAX sînt pătrate, iar cele corespunzînd lui MIN sînt cercuri. Valorile evaluate sînt date la punctele finale.

În mod normal arborele are o adîncime mare și pleacă mai multe arce din fiecare nod. Chiar pentru jocuri de complexitate medie este practic imposibilă generarea arborelui incluzînd toate punctele finale. În astfel de cazuri, valorile evaluate sînt găsite prin generarea unui arbore de adîncime rezonabilă, în care evaluarea punctelor terminale ale arborilor parțiali se face prin aceleași metode. Valorile determinate pe această cale sînt numite *valori evaluate static*. Dacă jocul este aproape de terminare se poate atinge un punct final, însă de cele mai multe ori cea mai bună mutare va fi decisă prin utilizarea valorilor evaluate static. Se ia în considerație faptul că adversarul determină și el cea mai bună mutare după generarea unui arbore parțial de adîncime

corespunzătoare. Adîncimea arborelui este restricționată prin memoria și timpul de calcul necesare. Valorile evaluate static sînt date de cunoașterea asupra caracteristicilor jocului. Pentru a juca șah, Shogi sau go este necesară atît investigarea metodelor de căutare cît și a metodelor de evaluare adecvate. În lucrare se prezintă numai metode de căutare atunci cînd valorile evaluate sînt considerate cunoscute.

▼ Fig. 7.9. Joc simplu reprezentat ca arbore



7.4.2. Metoda minimax

Se va explica metoda minimax cu referire la exemplul din fig. 7.9. De la nodul de start S , MAX are de ales între două posibilități. Dacă se alege primul arc se ajunge în nodul A . MIN are în continuare alte două posibilități, dar alegerea nodului D dă cea mai mică valoare evaluată. Dacă MAX a ales inițial nodul B , MIN va alege nodul E care dă valoarea inferențiată 1. MAX trebuie să aleagă propria mutare presupunînd că valoarea inferențiată pentru fiecare nod MIN este cea mai mică din valorile evaluate pentru toate nodurile pornind din el. Dacă se notează nodul MAX n și fii săi n_i ($i = 1, m$), iar nodul fiu al fiecăruia n_{ij} ($j = 1, i_m$), MAX maximizează valoarea evaluată a lui $f(n_i)$. Selectarea lui i va satisface următoarea condiție $f(n_i) = \max f(n_i)$. MIN minimizează $f(n_i)$ pentru un i dat și alege j astfel încît $f(n_i) = \min f(n_{ij})$. Așa că MAX alege i în concordanță cu:

$$f(n_i) = \max\{\min f(n_{ij})\}.$$

Dacă se ia adîncimea arborelui k , MAX va selecta i_1 așa că

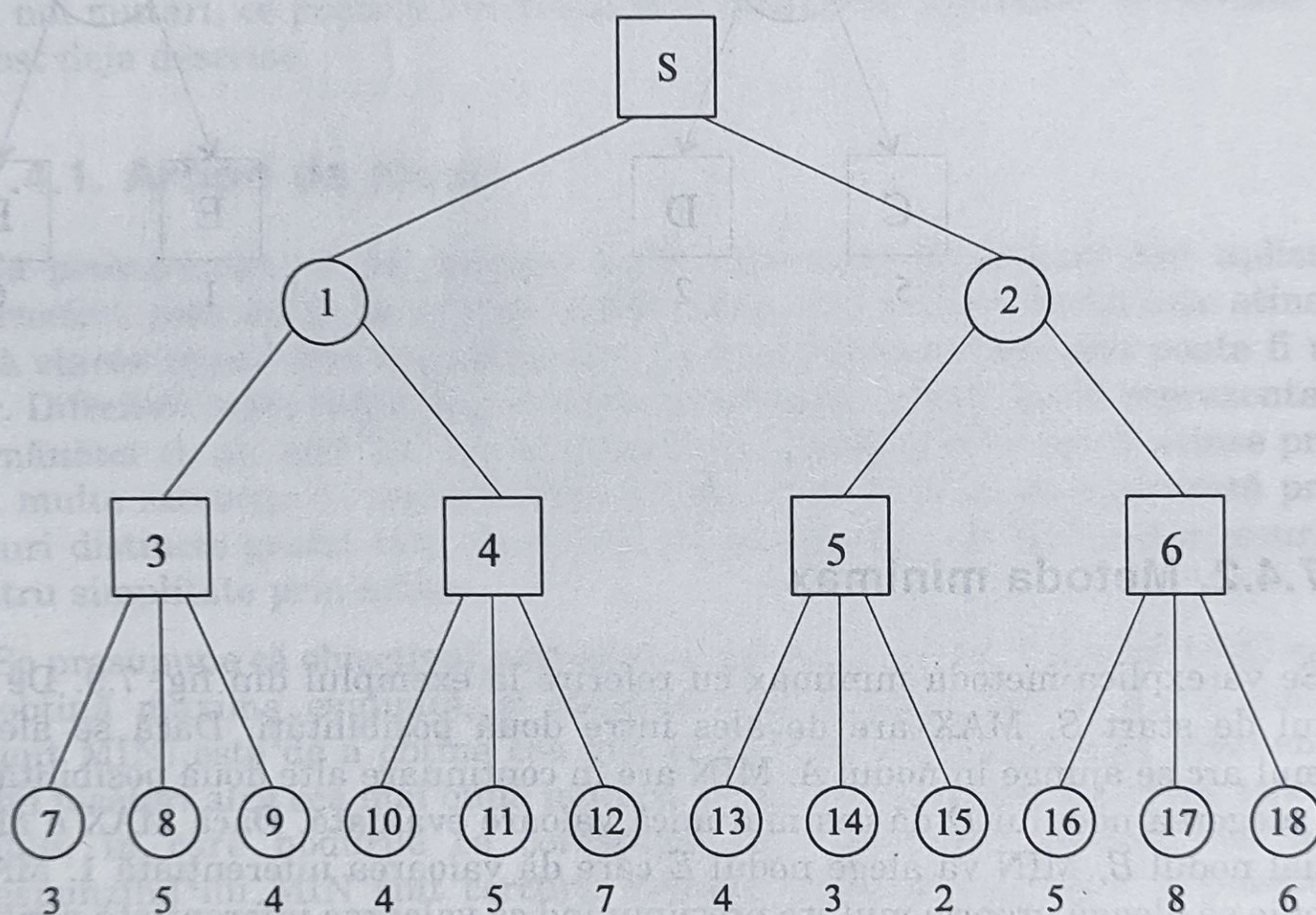
$$f(n_{i_1, i_2, \dots, i_k}) = \max \min \dots \{f(n_{i_1, i_2, \dots, i_k})\}$$

Dacă valorile evaluate ale punctelor terminale din arbore sînt corecte, metoda minimax va selecta cea mai bună mutare. Totuși, dacă adîncimea arborelui crește, numărul de noduri ce trebuiesc evaluate crește exponențial. Cercetările actuale caută să găsească cel mai eficient algoritm. Un exemplu tipic este explicat în paragraful 7.4.3.

7.4.3. Metoda alfa-beta

Fie arborele de căutare din fig. 7.10, în care nodurile sînt numerotate și un scor este dat la fiecare nod terminal. Dacă se găsește cele mai bune mutări utilizînd algoritmul minimax, inițial MAX alege nodul 1, MIN alege nodul 3, MAX alege nodul 8 ce are valoare evaluată 5.

▼ Fig. 7.10. Căutarea în arbori utilizînd metoda alfa-beta



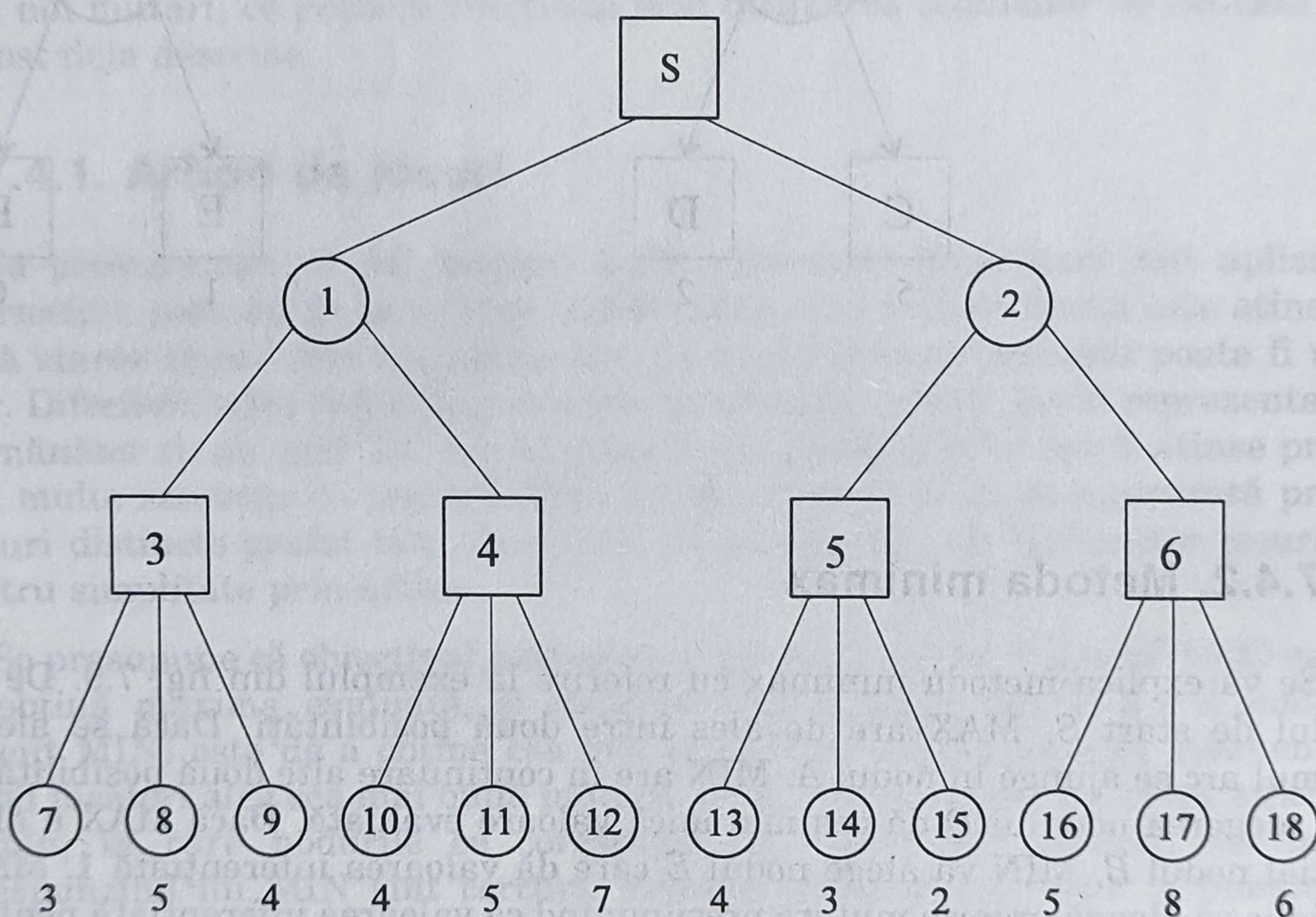
Presupunînd că într-o procedură ce este similară cu cea de căutare în adîncime, s-a obținut valoarea evaluată 5 pentru nodul 3, în continuare se încearcă găsirea valorii nodului 4. Se cunoaște la aceste momente faptul că valoarea nodului 1 nu poate fi mai mare decît 5. Motivul rezidă în faptul că dacă valoarea nodului 4 ar fi mai mare decît 5, MIN nu va alege nodul 4. Cu toate că se cunoaște că, valorile nodurilor 10 și 11 sînt obținute, valoarea nodului 4 este cel puțin 5. Așa că nu poate fi mai avantajos pentru MIN să aleagă nodul 4 în detrimentul nodului 3. Se deduce că MIN va alege nodul 3

Dacă valorile evaluate ale punctelor terminale din arbore sînt corecte, metoda minimax va selecta cea mai bună mutare. Totuși, dacă adîncimea arborelui crește, numărul de noduri ce trebuie evaluate crește exponențial. Cercetările actuale caută să găsească cel mai eficient algoritm. Un exemplu tipic este explicat în paragraful 7.4.3.

7.4.3. Metoda alfa-beta

Fie arborele de căutare din fig. 7.10, în care nodurile sînt numerotate și un scor este dat la fiecare nod terminal. Dacă se găsește cele mai bune mutări utilizînd algoritmul minimax, inițial MAX alege nodul 1, MIN alege nodul 3, MAX alege nodul 8 ce are valoare evaluată 5.

▼ Fig. 7.10. Căutarea în arbori utilizînd metoda alfa-beta



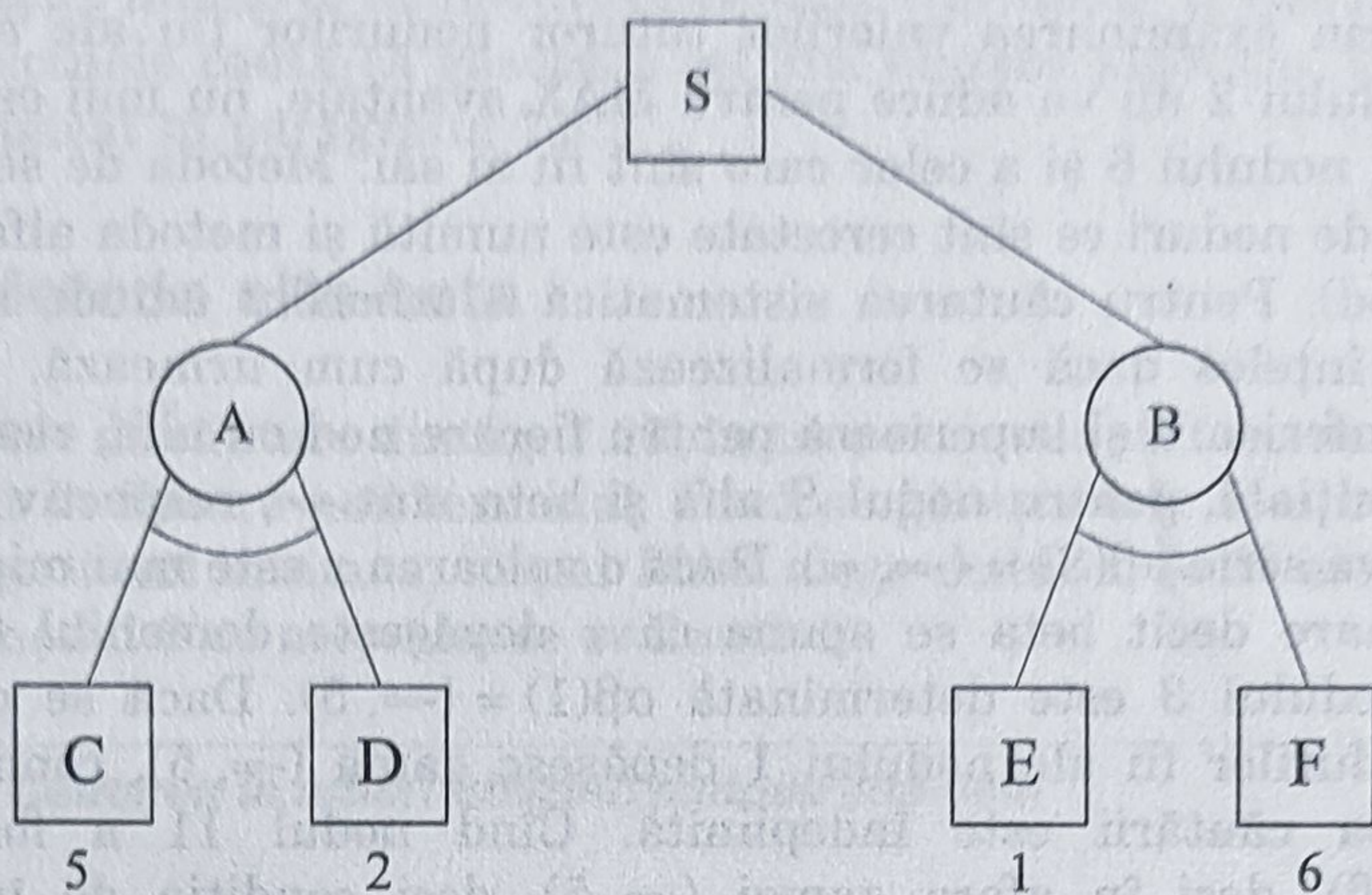
Presupunînd că într-o procedură ce este similară cu cea de căutare în adîncime, s-a obținut valoarea evaluată 5 pentru nodul 3, în continuare se încearcă găsirea valorii nodului 4. Se cunoaște la aceste moment faptul că valoarea nodului 1 nu poate fi mai mare decît 5. Motivul rezidă în faptul că dacă valoarea nodului 4 ar fi mai mare decît 5, MIN nu va alege nodul 4. Cu toate că se cunoaște că, valorile nodurilor 10 și 11 sînt obținute, valoarea nodului 4 este cel puțin 5. Așa că nu poate fi mai avantajos pentru MIN să aleagă nodul 4 în detrimentul nodului 3. Se deduce că MIN va alege nodul 3

fără să mai examineze nodul 12. În acest mod valoarea ce determină costul nodului 1 este 5 și ca urmare valoarea lui S trebuie să fie cel puțin 5. Considerînd în acest moment că valoarea evaluată 4, este obținută pentru nodul 5, prin examinarea valorilor tuturor nodurilor fiu ale sale, și cum selecția nodului 2 nu va aduce pentru MAX avantaje, nu mai este necesară inspectarea nodului 6 și a celor care sînt fii ai săi. Metoda de segmentare a numărului de noduri ce sînt cercetate este numită și metoda alfa-beta (alfa-beta method). Pentru căutarea sistematică a arborilor adînci, metoda este simplă de înțeles dacă se formalizează după cum urmează. Se notează marginea inferioară și superioară pentru fiecare nod cu alfa, respectiv beta. În starea inițială, pentru nodul S alfa și beta sînt $-\infty$, respectiv $+\infty$. Pentru aceasta se va scrie $\alpha\beta(S) = (-\infty, \infty)$. Dacă o valoare x este mai mică decît alfa sau mai mare decît beta se spune că x depășește domeniul (α, β) . Cînd valoarea nodului 3 este determinată $\alpha\beta(1) = (-\infty, 5)$. Dacă se cunoaște că valorile nodurilor fii ale nodului 1 depășesc gama $(-\infty, 5)$, condiția pentru întreruperea căutării este îndeplinită. Cînd nodul 11 a fost explorat $\alpha\beta(4) = (5, \beta)$, deci în afara gamei $(-\infty, 5)$, deci condiția de întreruperea căutării este din nou satisfăcută și nu va fi necesară căutarea în continuare a fiilor lui 4. La acest moment $\alpha\beta(1) = (5, 5)$ și $\alpha\beta(5) = (5, \infty)$. Cînd căutarea continuă valoarea nodului 5 este găsită ca fiind 4, în afara gamei $(5, \infty)$ și se întrerupe căutarea nodului 2 și a celor ce pornesc din el.

Prima întrerupere a căutării apare datorită faptului că nodul 4 este cel puțin egal cu valoarea lui beta și a părinților săi, a doua întrucît valoarea nodului 2 este cel puțin egală cu valoarea lui alfa. Procedul este numit beta-cut, respectiv alfa-cut. Metoda de creștere a eficienței căutării utilizînd marginile inferioare și superioare este numită metoda alfa-beta. Pentru a determina eficiența căutării se calculează numărul maxim al nodurilor explorate în cel mai bun caz. Considerînd momentul în care s-a găsit valoarea k a primului nod fiu de la un nod MAX, nodul n . $\alpha\beta(n) = (k, \infty)$. Dacă beta al nodului părinte m al lui n este mai mic sau egal cu k , nu este necesară evaluarea celorlalte noduri fiu ale lui n . Dacă nodul m nu este primul nod fiu al părintelui său, valoarea beta este deja dată și nu poate fi ∞ . Concluzii similare pot fi trase pentru nodurile MIN. În consecință, în cel mai bun caz, dacă nodul m nu este primul nod al unui nod părinte, nu este necesară examinarea nici unui nod fiu decît a primului nod al lui m .

Nodurile terminale sînt reprezentate prin secvența de arce ce le leagă. De exemplu nodul 12 din fig. 7.10 poate fi exprimat prin secvența (1, 2, 3), adică este atins urmărind primul arc al lui S , al doilea de la nodul 1 și al treilea de la nodul 4. Ca urmare nodurile terminale ale arborelui de adîncime k se exprimă prin k -uplul (e_1, e_2, \dots, e_k) . Nodurile ce nu necesită examinarea sînt cele pentru care $e_1 \neq 1, e_{i+1} \neq 1$. În consecință nodurile care au fost examinate în cel mai bun caz, fie sînt noduri numere pare care au în secvență 1, fie sînt noduri numere impare care au în secvență 1. Noțiunea de par sau impar se referă la numărul nodului în adîncimea arborelui.

▼ Fig. 7.11. Arbore ȘI/SAU pentru jocuri



Dacă se presupune că numărul arcelor posibile de la noduri neterminale este m , atunci numărul nodurilor pare $N_0(k, m)$ și numărul nodurilor impare $N_e(k, m)$ sînt date prin următoarele formule:

dacă k este par $N_0(k, m) = m^{(k-1)/2}$, $N_e(k, m) = m^{(k+1)/2}$

dacă k este impar $N_0(k, m) = m^{k/2}$, $N_e(k, m) = m^{k/2}$

Nodul $(1, 1, \dots, 1)$ este numărat pentru ambele $N_0(k, m)$ și $N_e(k, m)$, așa că $N_{\text{total}}(k, m)$ este $N_0(k, m) + N_e(k, m) - 1$. Cu aceste precizări se obține

pentru k par $N_{\text{total}}(k, m) = m^{(k-1)/2} + m^{(k+1)/2} - 1$

pentru k impar $N_{\text{total}}(k, m) = 2m^{k/2} - 1$.

În cel mai bun caz, numărul nodurilor ce trebuie examinate prin metoda alfa-beta este aproximativ același cu numărul nodurilor ce trebuie căutat prin metoda minimax într-un arbore de adîncime jumătate. În cazul cel mai defavorabil toate nodurile vor trebui examinate. Uzual situația este cuprinsă între două extreme, valoarea reală fiind dată prin inegalitatea

$$m^{k/2} < N(k, m) \leq m^k$$

7.4.4. Utilizarea metodei de căutare în grafuri ȘI/SAU

Un arbore specific pentru jocuri este reprezentat ca un graf ȘI/SAU. De exemplu arborele din fig. 7.9 reprezentat ca un graf este arătat în fig. 7.11. Nodurile fiu ale nodurilor MAX corespund la noduri SAU, cele ale nodurilor MIN la noduri ȘI (nodul de start este văzut ca un nod ȘI). Ca și la arborii ȘI/SAU nodurile terminale au valoarea evaluată dată. Costul arcelor este întotdeauna 0. Utilizînd metoda minimax, cea mai bună mutare este determinată prin următoarele metode:

- ♦ dacă nodurile fiu ale unui nod dat sînt noduri ȘI valoarea nodului este dată de cea mai mică valoare evaluată a fiilor săi;
- ♦ dacă nodurile fiu sînt noduri SAU valoarea este dată de valoarea maximă dintre valorile nodurilor fiu ale sale.

În cazul jocurilor pentru care obiectivul este cel de a face valoarea nodului de start maximă, valorile evaluate sînt în opoziție față de cost, adică cea mai mare valoare evaluată este cea mai bună. În consecință, metoda de calcul a valorilor în diverse noduri ale grafului este diferită de cea utilizată în metodele descrise anterior pentru arbori ȘI/SAU. Valoarea evaluată a nodului terminal a soluției grafului este însăși valoarea nodului terminal. Valoarea într-un nod unde fii sînt noduri ȘI este valoarea minimă din valorile asociate nodurilor fiu. Valoarea unui nod unde fii sînt noduri SAU este aceeași cu valoarea nodului fiu. Dacă se scrie valoarea nodului n ca $f(n)$, $f(A) = 2$ și $f(S) = 2$. Alte soluții pot apare din valorile celorlalte noduri. În cazul de față soluția este 1. Metoda descrisă în capitolul anterior referitoare la căutarea în grafuri ȘI/SAU poate fi aplicată și la arbori de jocuri. Totuși, $h'(f')$ nu este dată de aceeași formulă exceptînd nodurile terminale. Pentru a obține soluția optimă condiția $f'(n) \geq f(n)$ este îndeplinită pentru toate nodurile (cînd se urmărește obținerea soluției de cost minim $h'(n) \leq h(n)$ pentru toate nodurile). În consecință dacă valorile punctelor terminale nu sînt date, valorile inferențiate ale arborilor candidat sînt infinit de mari. Presupunînd că, la expandarea nodurilor neterminale ale grafului candidat, nodurile fiu sînt noduri ȘI ele se expandează la același moment. Dacă se notează primul nod fiu cu n_1 (nod ȘI) al lui n , se poate vedea că la momentul în care valoarea lui n_i este găsită, $f(n) \leq f(n_1)$. Ca urmare se poate lua $f'(n) \leq f(n_1)$. Așa se poate calcula f' și reactualiza în timpul procesului de căutare. Asemănător cu metoda de căutare optimală descrisă în capitolul precedent, principiul acestei metode este bazat pe faptul că un graf candidat adus din open_list va fi expandat după cea mai mare valoare inferențiată.

În arbori ȘI/SAU pentru jocuri găsirea soluției necesită nu numai nodul terminal care dă valoarea soluției arborelui ci și drumul urmat pentru ca pornind de la nodul de start acesta să fie atins. Pentru transpunerea algoritmului, prin convenție se va nota cu EXPL nodurile atinse ce nu au starea SOLVED. În acest context starea nodului n este dată de tripletul (n, s, v) , în care s indică faptul că nodul n este SOLVED sau EXPL, v dă valoarea lui $f(n)$ dacă n este SOLVED sau $f'(n)$ dacă el este EXPL. Starea nodului n este pusă în open_list printr-o procedură pe care am notat-o putopen (n, s, v) . De asemenea se va scrie pentru nodul părinte parent(n), nodul terminal n ca terminal(n) și pentru a detecta un nod n ca fiind ȘI/SAU s-a utilizat funcția type(n). Se dă în pseudocod algoritmul de căutare în astfel de situații.

```

Putopen (S, EXPL.    );
while true
{
  if open = goală then exit (fail);
  n = first (open);
  if (n = S & solved (n)) then exit (succes);

```



```

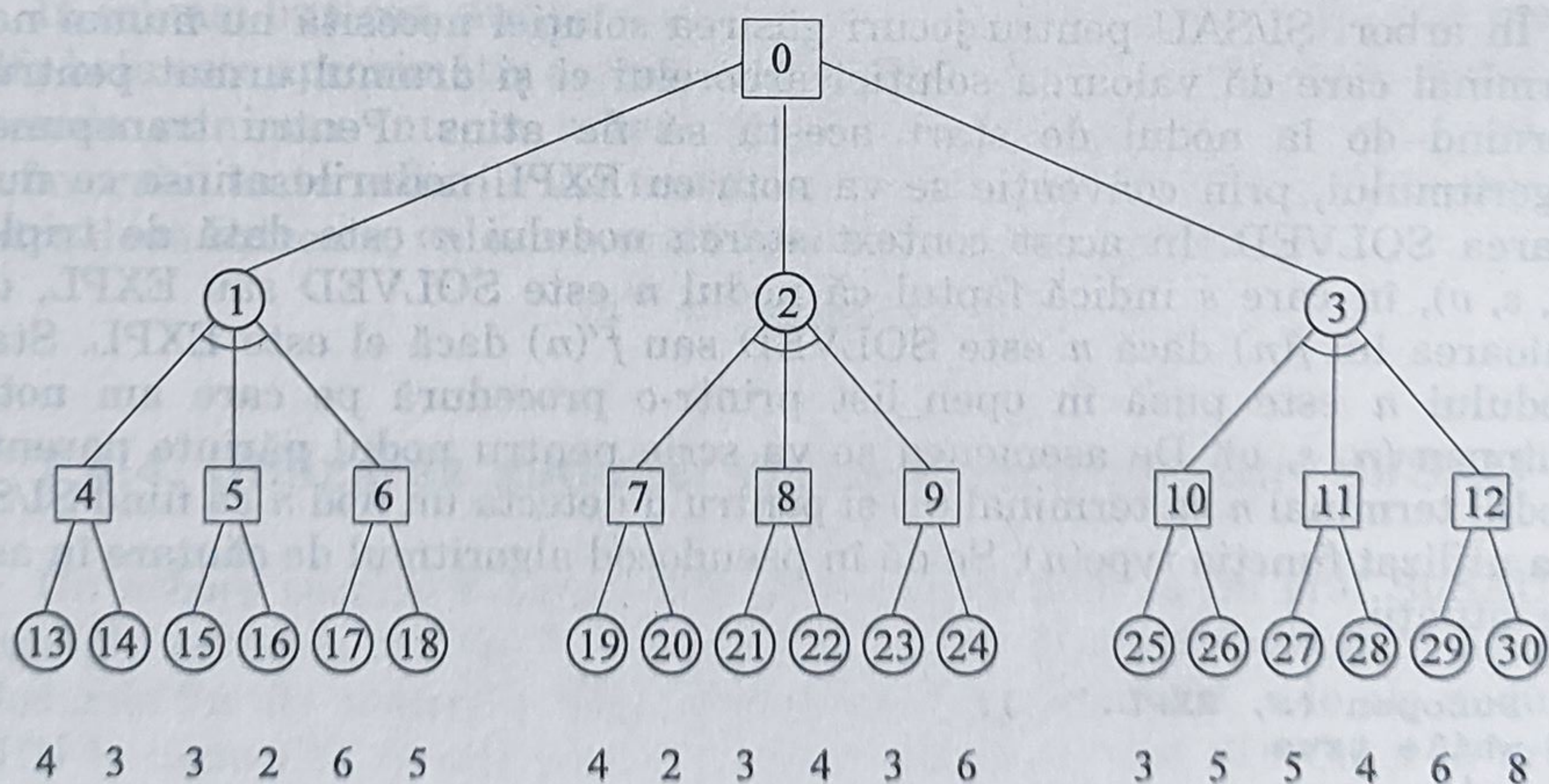
remove (n, open);
/*1*/ if ((s = EXPL) & (type(n) = AND) & not_terminal (n)) then
    pentru toate nodurile fiu n, ale lui n putopen (n, EXPL,
        f'(n));
/*2*/ if ((s = EXPL) & (type(n) = OR) & not_terminal (n))
    {next_n = next_fiu ( )
    if (next_n = SOLVED)
        putopen (n, SOLVED, f'(n));
    else putopen (n, EXPL, f'(n));}
/*3*/ if ((s = EXPL) & terminal (n))
    putopen (n, SOLVED, min (f'(n), f(n)));
/*4*/ if ((s = SOLVED) & (type(n) = AND))
    putopen (parent(n), EXPL, f'(n));
/*5*/ if ((s = SOLVED) & (type(n) = OR))
    {m = parent(n);
    mută toate nodurile fiu ale lui m în open_list
    putopen (n, SOLVED, f'(n));}
}

```

Pentru exemplificarea metodei se consideră graful din fig. 7.12. Așa cum se vede din rezultatele indicate în fig. 7.13 metoda alfa-beta este dependentă de ordinea nodurilor. În cazul de față dacă nodul 3 ar fi în locul nodului 1 soluția optimă este găsită mult mai rapid. În algoritm au fost notate cele 5 situații ce sînt ilustrate și în fig. 7.13 pentru a ușura cititorul în înțelegerea algoritmului.

Spre deosebire de căutarea optimală obișnuită care explorează mai întâi cele mai promițătoare noduri și a cărei eficiență nu este afectată de aranjarea nodurilor, metoda alfa-beta este dependentă de aceasta. În al doilea rînd memoria necesară pentru algoritmul alfa-beta este mult mai redusă. Alegerea uneia din cele două metode este dependentă de performanțele dorite din punctul de vedere al vitezei de calcul și al memoriei necesare.

▼ Fig. 7.12. Căutarea soluției optime în arbore de joc ȘI/SAU



▼ Fig. 7.13. Conținutul *open_list* și procedura utilizată

Pas	Proc	<i>open_list</i> în care <i>S</i> este SOLVED, <i>E</i> este EXPL
1		$(0, E, \infty)$
2	1	$(1, E, \infty), (2, E, \infty), (3, E, \infty)$
3	2	$(4, E, \infty), (2, E, \infty), (3, E, \infty)$
4	1	$(13, E, \infty), (14, E, \infty), (2, E, \infty), (3, E, \infty)$
5	3	$(14, E, \infty), (2, E, \infty), (3, E, \infty), (13, S, 4)$
6	3	$(2, E, \infty), (3, E, \infty), (13, S, 4), (14, S, 4)$
7	2	$(7, E, \infty), (3, E, \infty), (13, S, 4), (14, S, 3)$
8	1	$(19, E, \infty), (20, E, \infty), (3, E, \infty), (13, S, 4), (14, S, 3)$
9	3	$(20, E, \infty), (3, E, \infty), (13, S, 4), (19, S, 4), (14, S, 3)$
10	3	$(3, E, \infty), (13, S, 4), (19, S, 4), (14, S, 3), (20, S, 2)$
11	2	$(10, E, \infty), (13, S, 4), (19, S, 4), (14, S, 3), (20, S, 2)$
12	1	$(25, E, \infty), (26, E, \infty), (13, S, 4), (19, S, 4), (14, S, 3), (20, S, 2)$
13	3	$(26, E, \infty), (13, S, 4), (19, S, 4), (14, S, 3), (25, S, 3), (20, S, 2)$
14	3	$(26, S, 5), (13, S, 4), (19, S, 4), (14, S, 3), (25, S, 3), (20, S, 2)$
15	5	$(10, S, 5), (13, S, 4), (19, S, 4), (14, S, 3), (20, S, 2)$
16	4	$(3, E, 5), (13, S, 4), (19, S, 4), (14, S, 3), (20, S, 2)$
28	5	$(S, S, 5), (13, S, 4), (19, S, 4), (4, S, 3), (20, S, 2)$

CONTROLUL REZOLVĂRII PROBLEMELOR

În modul de rezolvare a problemelor s-a folosit reprezentarea în spațiul stărilor. Cunoașterea utilizată pentru realizarea unei căutări eficiente este dată sub forma costului stărilor specificate, a costului obiectivului, sau a costului de la starea inițială la o stare specificată. Când problema devine complexă este dificil să se reducă aceasta la o simplă problemă de căutare. Cum se determină strategia de rezolvare a problemei este o chestiune importantă. În acest capitol se tratează metodele de rezolvare a problemelor complicate utilizând planuri de rezolvare.

8.1. General problem solver (GPS)

GPS este un model al modului de rezolvare a problemelor de către oameni ce a fost introdus de filozofii A. Newell, J. C. Shaw, și H. A. Simion. Metoda utilizează un schelet neparticuliar pentru o problemă dată, însă pentru rezolvarea unei probleme arbitrare utilizează cunoașterea dată pentru problemă. Întrucât scopul este de a introduce cititorul în metoda GPS nu se tratează decât algoritmul specific GPS. Formalizarea GPS necesită introducerea conceptului de procedură recursivă, procedura care se utilizează pe ea însăși în propria definiție. Procedura ce se utilizează pe ea însăși în conversia stării S într-o altă stare G este:

```
Procedura recursivă GPS ( $S$ ,  $G$ )
If  $S$  include  $G$  then return ( $S$ ); /* dacă starea  $S$  satisface starea
     $G$  atunci obiectivul este atins */
Caută diferența între  $G$  și  $S$ . Depune aceasta într-o listă numită
    difference_list ordonată în ordinea importanței;
while true
{
    if stare (difference_list) = vidă then return (eșec);
     $d$  = first (difference_list);
    elimină ( $d$ , difference_list); /* elimină articolul extras din
        difference_list */
    Găsește toți operatorii care reduc diferența  $d$  și îi plasează
        în operator_list;
    while (stare (operator_list) # vidă)
```



```

{
  op = first (difference_list);
  elimină (op, operator_list);
  pc = condițiile operatorului;
   $S_1$  = GPS ( $S$ , pc); /*  $S_1$  este o nouă stare după aplicarea
    procedurii GPS ( $S$ , pc) */
  If ( $S_1$  = eșec) then break;
   $S_2$  = op ( $S_1$ ); /* aplică op asupra lui  $S_1$  */
   $S_3$  = GPS ( $S_2$ ,  $G$ );
  If ( $S_3$  = eșec) then break;
}
}
return ( $S_3$ ).

```

Dacă în problemă se cere schimbarea stării inițiale S_0 în starea finală G_0 , se rezolvă prin apelarea procedurii recursive GPS (S_0 , G_0). Pentru aceasta GPS divide obiectivul de atingerea lui G într-un subobiectiv al lanțului de la starea S la starea S_1 satisfăcând condițiile pc, apoi subscopul de atingerea stării S_2 prin aplicarea operatorilor asupra stării S_1 , și așa mai departe.

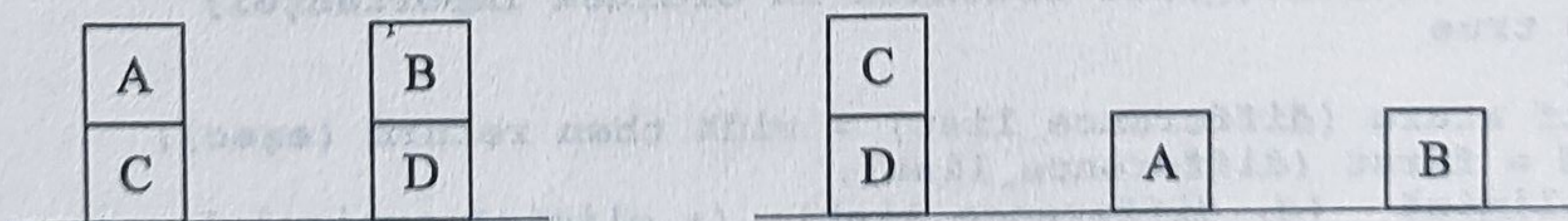
8.2. Planuri

Cînd procedura pentru rezolvarea unei probleme date devine foarte complicată o metodă eficientă este cea de stabilire a unui plan general pentru obținerea soluției. Așa cum s-a menționat mai sus, o metodă este bazată pe reducerea celei mai importante diferențe între starea curentă și starea obiectiv. În acest paragraf se tratează modul în care se formalizează planul pentru rezolvarea problemelor de așezare a cutiilor. Mai întîi se descriu metodele de reprezentare a problemelor și metodele de căutare pentru obținerea soluțiilor, modul în care acestea sînt îmbunătățite prin utilizarea planului.

8.2.1. Reprezentarea problemelor și metode de soluționare de bază

Rezolvarea unei probleme constă în modificarea unei stări date numită și stare inițială într-o stare finală numită și stare obiectiv. O metodă de reprezentare a stării este și calculul cu predicate, situație în care sînt definiți o serie de operatori, iar pentru fiecare operator modul în care starea se modifică și care sînt condițiile pentru ca aceasta să fie aplicabil. Ca exemplul se consideră problema din fig. 8.1.

▼ Fig. 8.1. Problema simplă de aranjare cuburi



Starea inițială:

EMPTY; brațul robotului nu ține nimic
 ONTABLE (C); blocul C se găsește pe masă
 ON (A, C); A este deasupra lui C
 ONTABLE (D)
 ON (B, D)

Starea obiectiv:

ON (C, D)
 ONTABLE (D)
 ONTABLE (A)
 ONTABLE (B)

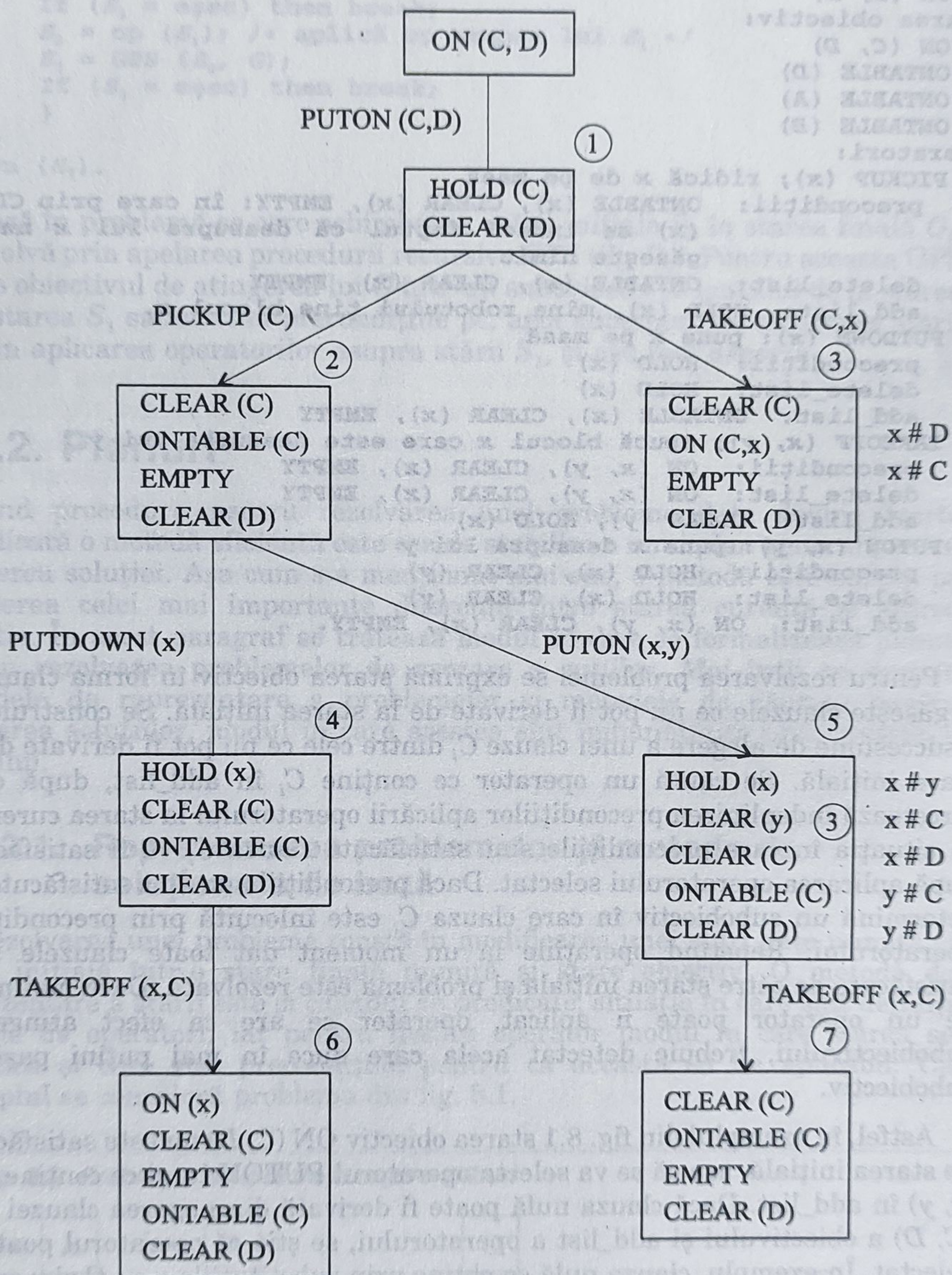
Operatori:

PICKUP (x); ridică x de pe masă
 precondiții: ONTABLE (x), CLEAR (x), EMPTY: în care prin CLEAR (x) se indică faptul că deasupra lui x nu se găsește nimic
 delete_list: ONTABLE (x), CLEAR (x), EMPTY
 add_list: HOLD (x), mîna robotului ține blocul x.
 PUTDOWN (x): pune x pe masă
 precondiții: HOLD (x)
 delete_list: HOLD (x)
 add_list: ONTABLE (x), CLEAR (x), EMPTY
 TAKEOFF (x, y); apucă blocul x care este deasupra lui y
 precondiții: ON (x, y), CLEAR (x), EMPTY
 delete_list: ON (x, y), CLEAR (x), EMPTY
 add_list: CLEAR (y), HOLD (x)
 PUTON (x, y): pune x deasupra lui y
 precondiții: HOLD (x), CLEAR (y)
 delete_list: HOLD (x), CLEAR (y)
 add_list: ON (x, y), CLEAR (x), EMPTY.

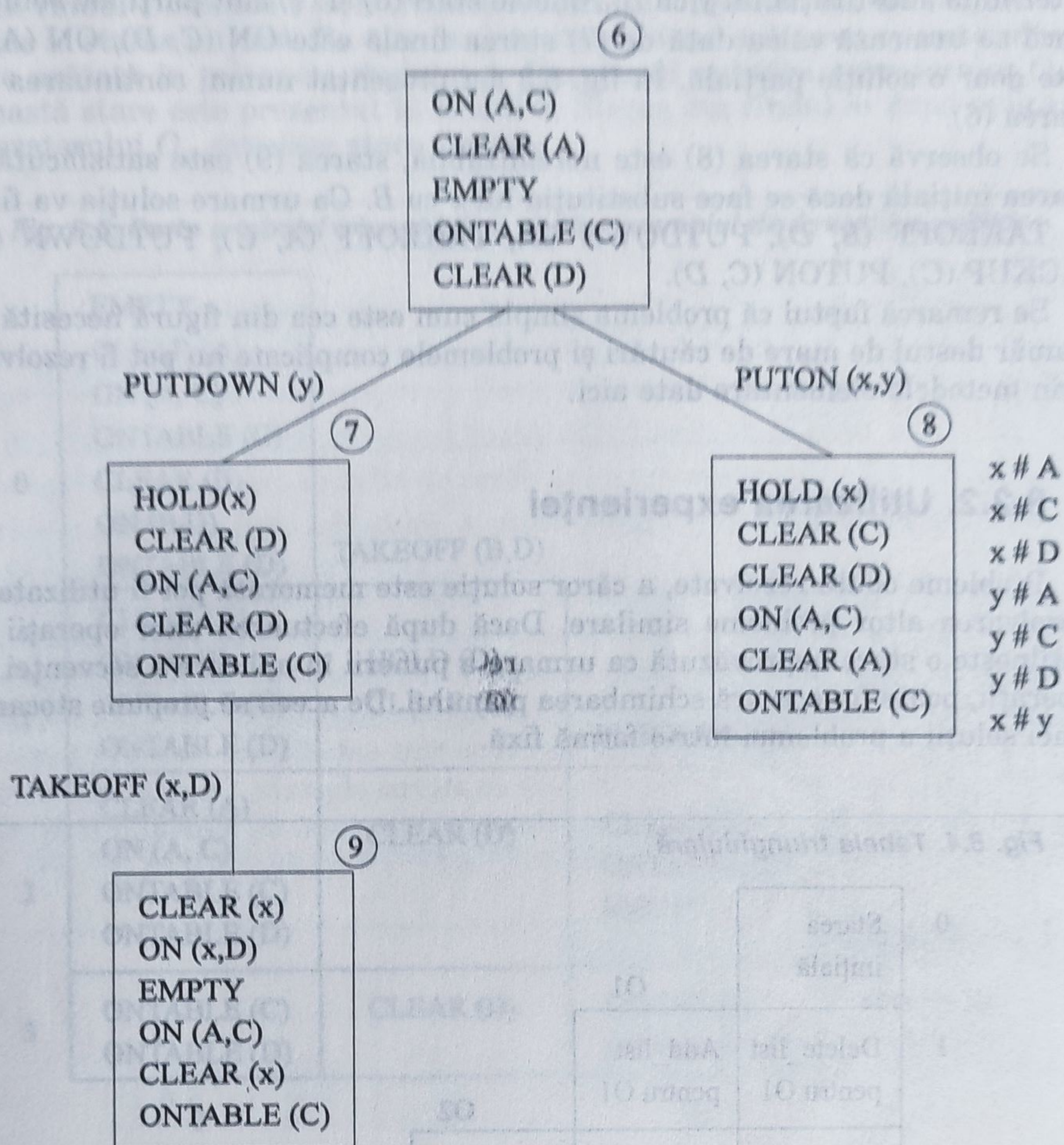
Pentru rezolvarea problemei se exprimă starea obiectiv în forma clauzală și găsește clauzele ce nu pot fi derivate de la starea inițială. Se construiește o succesiune de alegere a unei clauze C_i dintre cele ce nu pot fi derivate de la stare inițială. Se caută un operator ce conține C_i în add_list, după care cercetează îndeplinirea precondițiilor aplicării operatorului la starea curentă. În situația în care precondițiile sînt satisfăcute clauza C_i va fi satisfăcută după aplicarea operatorului selectat. Dacă precondițiile nu sînt satisfăcute se determină un subobiectiv în care clauza C_i este înlocuită prin precondițiile operatorului. Repetînd operațiile la un moment dat toate clauzele sînt satisfăcute de către starea inițială și problema este rezolvată. Dacă mai mult de un operator poate fi aplicat, operator ce are ca efect atingerea subobiectivului, trebuie detectat acela care duce în mai puțini pași la subobiectiv.

Astfel, în exemplul din fig. 8.1 starea obiectiv ON (C, D) nu este satisfăcută de starea inițială, așa că se va selecta operatorul PUTON (x, y) ce conține ON (x, y) în add_list. Dacă clauza nulă poate fi derivată din negarea clauzei ON (C, D) a obiectivului și add_list a operatorului, se știe că operatorul poate fi selectat. În exemplu, clauza nulă se obține prin substituțiile x cu C și y cu D. De aceea subobiectivul va trebui să satisfacă precondițiile operatorului PUTON (C, D), adică HOLD (C) și CLEAR (D). Mai departe, în funcție de clauzele HOLD (C) și CLEAR (D) se alege operatorul. Cînd un subobiectiv conține mai mult de o clauză chiar dacă un operator satisface una dintre

▼ Fig. 8.2. Prima parte a procesului de rezolvare din fig. 8.1



▼ Fig. 8.3. Continuarea procesului din starea (6)



clauze nu va fi selectat dacă alte clauze sînt inconsistente cu delete_list sau add_list a operatorului. De exemplu operatorul PUTDOWN (D) care are CLEAR (D) în add_list conține și EMPTY în add_list. Cum clauza este inconsistentă cu obiectivul HOLD(C) operatorul nu va fi selectat. Prima parte a soluției acestei probleme este dată în fig. 8.2. Restricțiile asupra lui x sînt aplicate în starea (3). Comparînd stările (2) și (3), (2) conține mai puține clauze ce nu sînt îndeplinite de starea inițială așa că este expandată. Clauzele din (2) care nu sînt îndeplinite de starea inițială sînt CLEAR (C) și CLEAR (D). Fiecare dintre operatori pot avea o prioritate dată. Clauzele din starea (6) în afară de CLEAR (D) sînt satisfăcute de starea inițială dacă se face substituția lui x cu A. În dreapta stării (5) sînt indicate restricțiile variabilelor.

Dacă se face substituția lui x cu D în (5) se obține o contradicție. Substituția lui x cu A este găsită în starea (7). Această substituție extinsă la (5) determină substituția lui y cu B . Ambele stări (6) și (7) sînt părți ale soluției. Dacă se urmează calea dată de (7) starea finală este ON (C, D), ON (A, B) este doar o soluție parțială. În fig. 8.3 s-a prezentat numai continuarea din starea (6).

Se observă că starea (8) este nerealizabilă, starea (9) este satisfăcută de starea inițială dacă se face substituția lui x cu B . Ca urmare soluția va fi:

TAKEOFF (B, D), PUTDOWN (B), TAKEOFF (A, C), PUTDOWN (A), PICKUP (C), PUTON (C, D).

Se remarcă faptul că problema simplă cum este cea din figură necesită un număr destul de mare de căutări și problemele complicate nu pot fi rezolvate prin metodele elementare date aici.

8.2.2. Utilizarea experienței

Probleme odată rezolvate, a căror soluție este memorată pot fi utilizate la rezolvarea altor probleme similare. Dacă după efectuarea unor operații se întâlnește o stare neprevăzută ca urmare a punerii în aplicare a secvenței de operații, poate fi necesară schimbarea planului. De aceea se propune stocarea unei soluții a problemei într-o formă fixă.

▼ Fig. 8.4. Tabela triunghiulară

0	Starea inițială	O1		
1	Delete_list pentru O1	Add_list pentru O1	O2	
	Delete_list pentru O2	Delete_list pentru O2	Add_list pentru O1	O3
				On
				Add_list pentru On

Forma fixă este numită tabelă triunghiulară și este prezentată în fig. 8.4. Starea inițială este introdusă în rîndul 0 coloana 0. Add_list obținută prin aplicarea operatorului O_1 asupra acesteia este introdusă în rîndul 1 coloana 1. În rîndul 1 coloana 0 se introduce rezultatul eliminării din delete_list a lui O_1 de la starea inițială. Cu alte cuvinte starea după aplicarea operatorului O_1 este arătată în coloanele rîndului 1. Rezultatul aplicării operatorului O_2 la această stare este prezentat în rîndul 2. Starea din rîndul m după aplicarea operatorului O_m satisface stare obiectiv.

▼ Fig. 8.5. Parte a tabelului triunghiular pentru exemplul de așezare a cutiilor

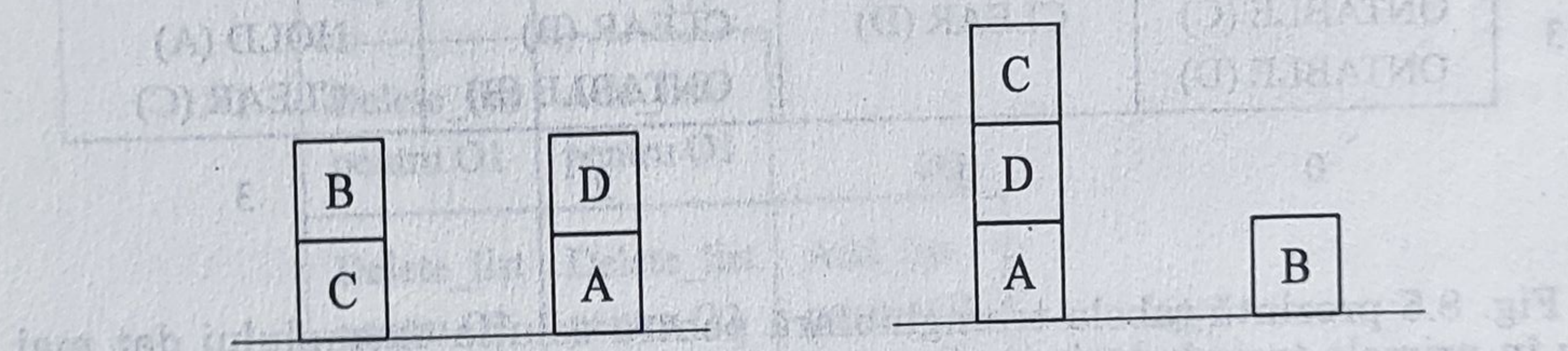
0	EMPTY			
	CLEAR (A)			
	ON (A, C)			
	ONTABLE (C)			
1	CLEAR (B)			
	ON (B,D)			
	ONTABLE (D)	TAKEOFF (B,D)		
2	CLEAR (A)			
	ON (A, C)			
	ONTABLE (C)	HOLD (D)		
	ONTABLE (D)	CLEAR (D)		
3	CLEAR (A)			
	ON (A, C)			
	ONTABLE (C)	CLEAR (D)	CLEAR (B)	
	ONTABLE (D)		ONTABLE (B)	
4			EMPTY	
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				
32				
33				
34				
35				
36				
37				
38				
39				
40				
41				
42				
43				
44				
45				
46				
47				
48				
49				
50				
51				
52				
53				
54				
55				
56				
57				
58				
59				
60				
61				
62				
63				
64				
65				
66				
67				
68				
69				
70				
71				
72				
73				
74				
75				
76				
77				
78				
79				
80				
81				
82				
83				
84				
85				
86				
87				
88				
89				
90				
91				
92				

▼ Fig. 8.6. Tabela triunghiulară generalizată

0	CLEAR (y) ON (y,v) EMPTY	TAKEOFF (y,v)					
1		HOLD (y)	PUTDOWN (y)				
2	CLEAR (x) ON (x,u)		EMPTY	TAKEOFF (x,v)			
3				HOLD (x)	PUTDOWN (x)		
4	ONTABLE(u)			CLEAR (u)	EMPTY	PICKUP (u)	
5						HOLD (v)	PUTON (u,v)
6		CLEAR (v)					ON (u,v)
	0	1	2	3	4	5	6

Se consideră un exemplu similar cu cel descris mai sus cu deosebirea că în obiectiv apare atît ON (C, D) cît și alte obiective și în plus diferă și starea inițială. Structura este prezentată în fig. 8.7. Se observă, în tabela din fig. 8.6 că dacă substituțiile u cu C , v cu D și x cu B sînt executate, porțiunea de sub și din stînga liniilor duble conține clauze ce sînt satisfăcute de stare inițială și deci starea din al șaselea rînd coincide cu starea obiectiv. Ca urmare, starea obiectiv poate fi atinsă prin secvența de operatori după O_3 TAKEOFF (B, C), PUTDOWN (B), PICKUP (C), PUTON (C, D).

▼ Fig. 8.7. Problemă de cuburi similară



Să considerăm acum situația cînd robotul întîlnește o stare neprevăzută la execuția operațiilor arătate în fig. 8.6. De exemplu cînd execută ultimul operator PUTON (C, D) el scapă blocul C. Presupunînd că robotul poate găsi blocul C utilizînd o cameră de luat vederi, starea satisface clauzele cuprinse la stînga coloanei 4 și sub rîndul 4, ceea ce face ca să înceapă cu PICKUP (C). Dacă se dispune de tabele triunghiulare pentru diferite operații, este posibilă mutarea dintr-o stare a unei tabele într-o altă stare din altă tabelă. Dacă clauzele din al i -lea rînd al tablei triunghiulare (incluzînd clauzele nemarcate), satisfac clauzele cuprinse în al i -lea rînd și la stînga coloanei j din

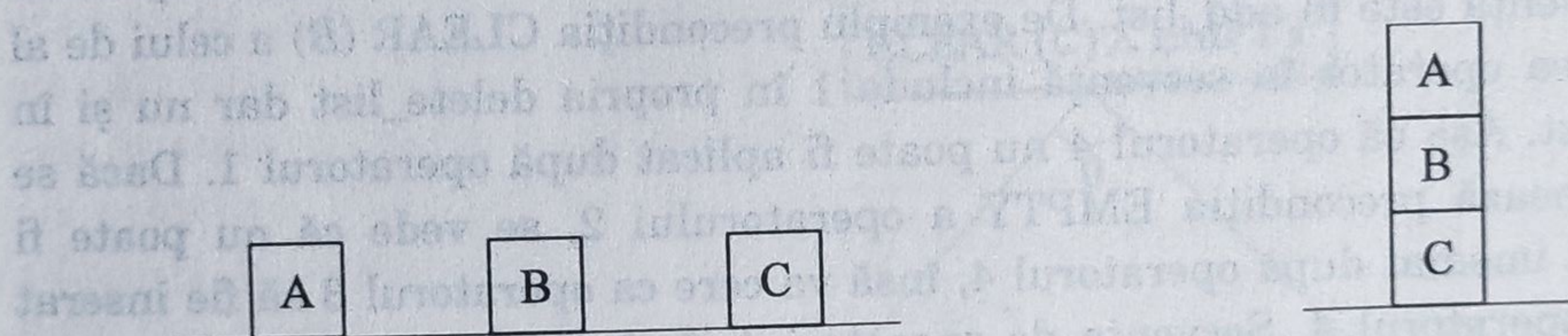
altă tabelă, după aplicarea operatorului O_i se poate muta la al $(j + 1)$ -lea operator al altei tabele. Astfel, dacă operații comparativ similare sînt executate la mai multe momente de timp, este uzual să se păstreze secvența operațiilor tipice în formă de tabele triunghiulare.

8.2.3. Inferențe cu mai multe obiective

La problemele anterioare s-a pus problema atingerii unui singur obiectiv, însă pe parcursul procesului atingerea subscopurilor include diferite clauze. Dacă aceste clauze sînt independente soluția problemei poate fi simplificată prin rezolvarea clauzelor separat. Dacă nu sînt independente, va trebui găsită o secvență de operații ce ating toate obiectivele în același timp. În acest paragraf se va explica metoda de rezolvare a diverselor obiective independent, precum și modul de obținere a unei soluții prin considerarea efectelor inferențelor mutuale între aceste obiective.

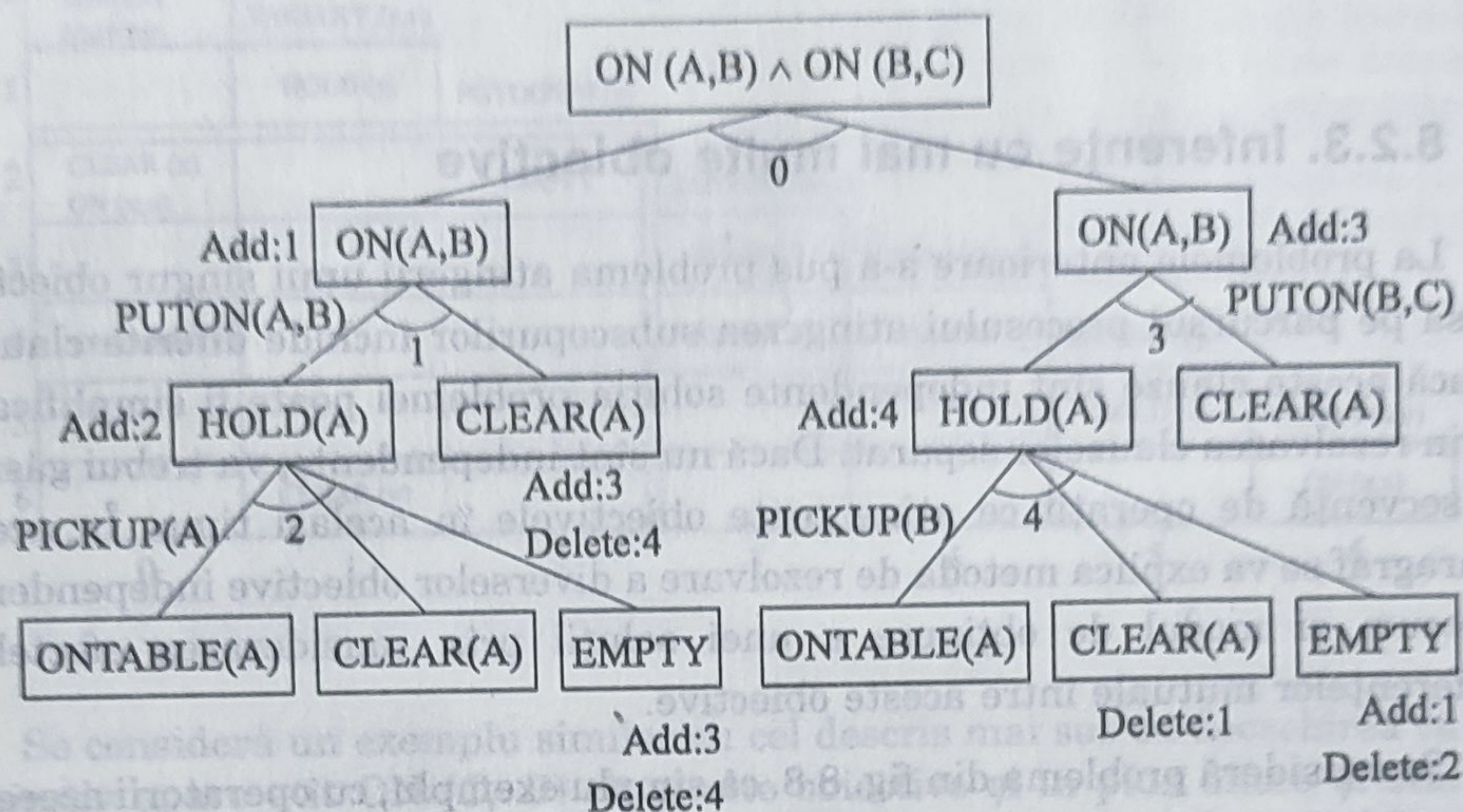
Se consideră problema din fig. 8.8. ca simplu exemplu, cu operatorii aceeași cu cei din exemplele anterioare. Problema este împărțită în diferite clauze ce sînt rezolvate, soluția problemei fiind reprezentată ca un arbore ȘI/SAU (fig. 8.9).

▼ Fig. 8.8. Exemplu de construire a turnului



Operatorii sînt denumiți prin nume arbitrare (separarea obiectivului inițial în două obiective este considerată ca operator). Două seturi de operatori sînt definite pentru fiecare obiectiv (add, delete). Operatorii add ai obiectivului C_i sînt setul operatorilor în arborele ȘI/SAU alții decât strămoșii lui C_i și care conțin C_i în propria add_list. Operatorii delete ai obiectivului C_i sînt setul operatorilor în arborele ȘI/SAU alții decât strămoșii lui C_i și care conțin C_i în propria delete_list. Ambele seturi sînt ilustrate în figură, numărul elementelor fiecărui set fiind 0 sau 1. Spre exemplu CLEAR (B) este adăugat prin operatorul PUTON (B, C) și este șters prin operatorul HOLD (B), acești operatori nefiind strămoși ai lui CLEAR (B).

▼ Fig. 8.9. Arborele ȘI/SAU al soluției cînd diferite obiective sînt considerate independent

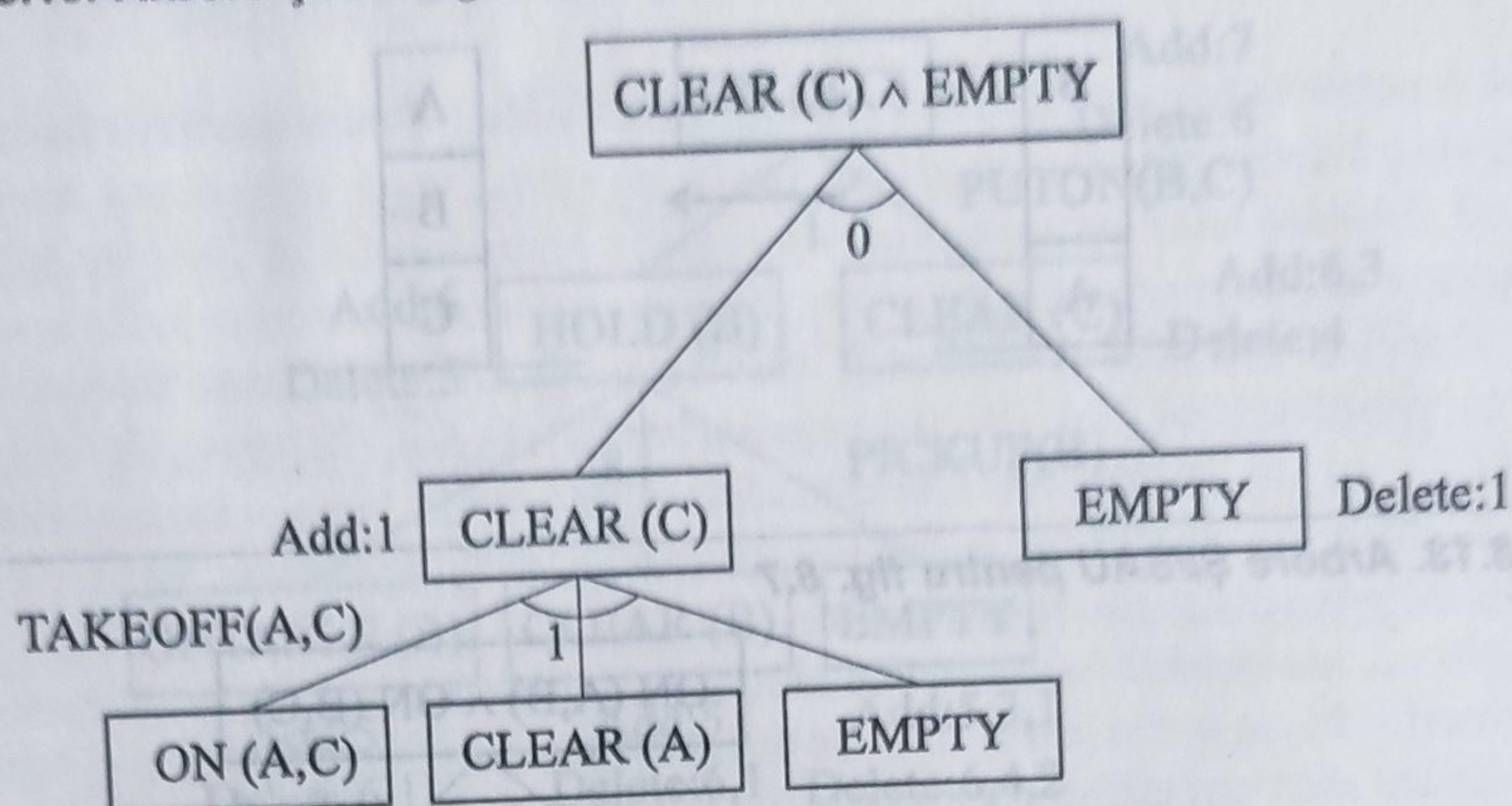


Restricțiile secvenței de operatori pot fi obținute din această figură. Pentru ca un operator O_i să fie aplicat, fie operatorii ce îl preced în secvență nu sînt incluși în add_list sau în delete_list a precondițiilor lui O_i , fie ultimul operator în secvență este în add_list. De exemplu precondiția CLEAR (B) a celui de al patrulea operator în secvență include 1 în propria delete_list dar nu și în add_list. Așa că operatorul 4 nu poate fi aplicat după operatorul 1. Dacă se examinează precondiția EMPTY a operatorului 2, se vede că nu poate fi aplicat imediat după operatorul 4, însă va cere ca operatorul 3 să fie inserat după operatorul 4. Secvența de operatori 4, 3, 2, 1 este de aceea o soluție candidată. Ea este o soluție completă întrucît satisface condițiile de aplicabilitate pentru toți operatorii.

8.2.4. Schimbarea planului

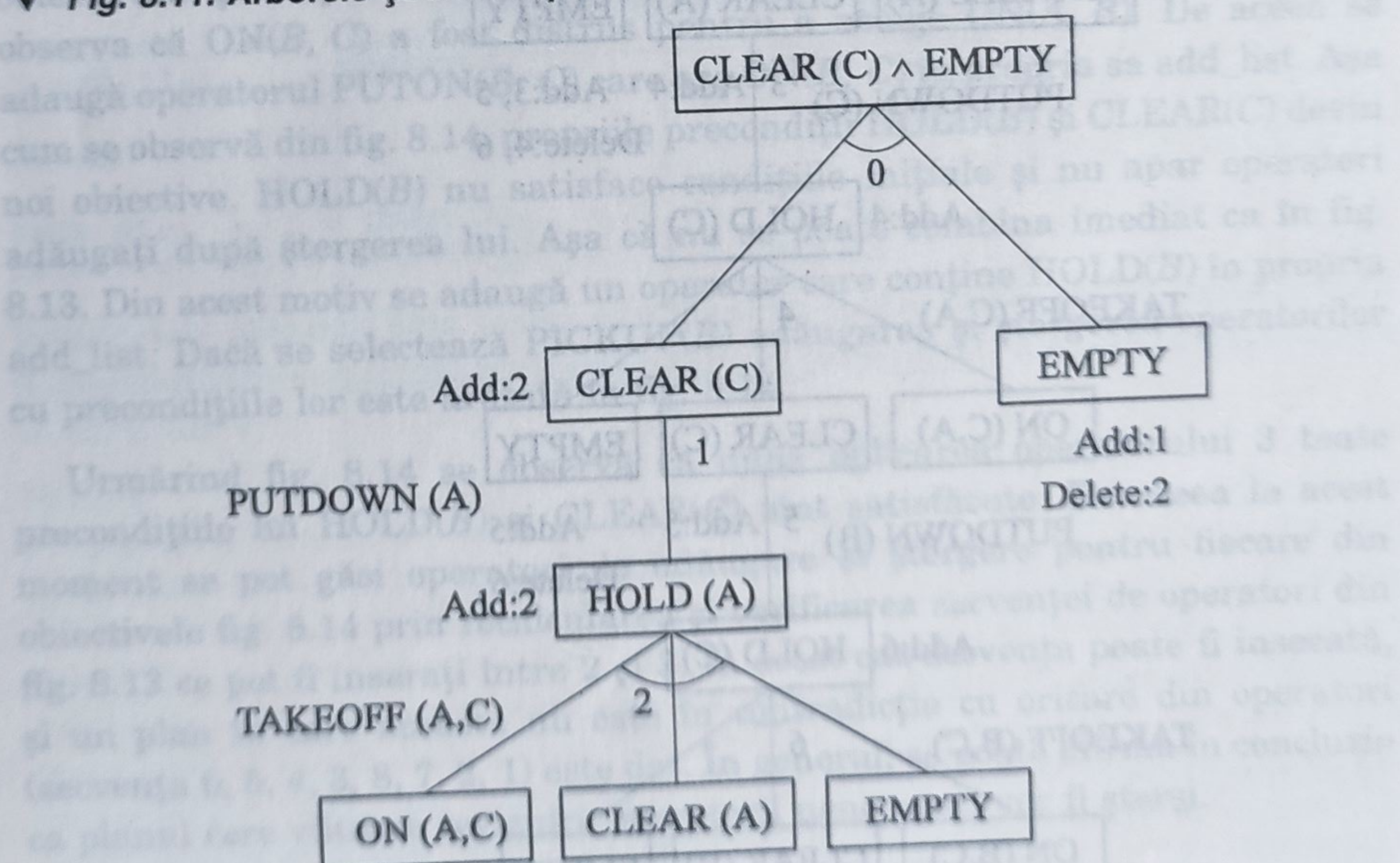
De multe ori problemele nu pot fi rezolvate prin separare și resintetizare. Ca urmare planul va fi schimbat prin adăugarea sau ștergerea operatorilor. Considerînd că starea (2) din exemplul prezentat în fig. 8.1, ONTABLE (C) este satisfăcut de starea inițială. Noul subobiectiv este acum atingerea simultană a precondițiilor CLEAR (C) și EMPTY. Dacă se împarte problema descrisă în secțiunea 8.2.3 se obține arborele din fig. 8.10.

▼ Fig. 8.10. Arbore ȘI/SAU generat în procesul de separare a problemei

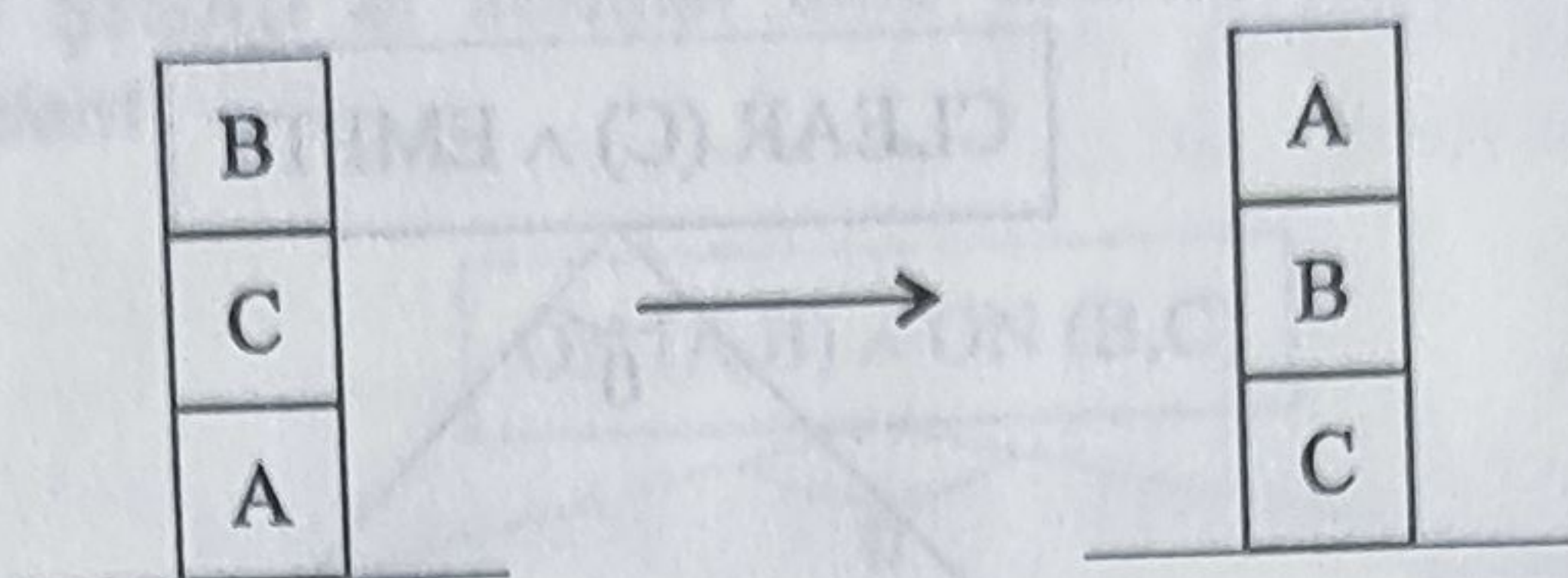


Scopul nu poate fi atins direct întrucât TAKEOFF(A, C) care este în add_list pentru CLEAR(C) este un operator de ștergere pentru EMPTY . Ca urmare, este nevoie de adăugarea unui operator care are EMPTY în propria add_list . Întrucât add_list pentru TAKEOFF(A, C) include HOLD(A) se caută un operator care convertește HOLD(A) în EMPTY . La selecția operatorului PUTDOWN(A) se obține soluția din fig. 8.11, metoda devenind mult mai eficientă față de găsirea secvenței de operatori prin căutare ca în paragraful 8.2.1.

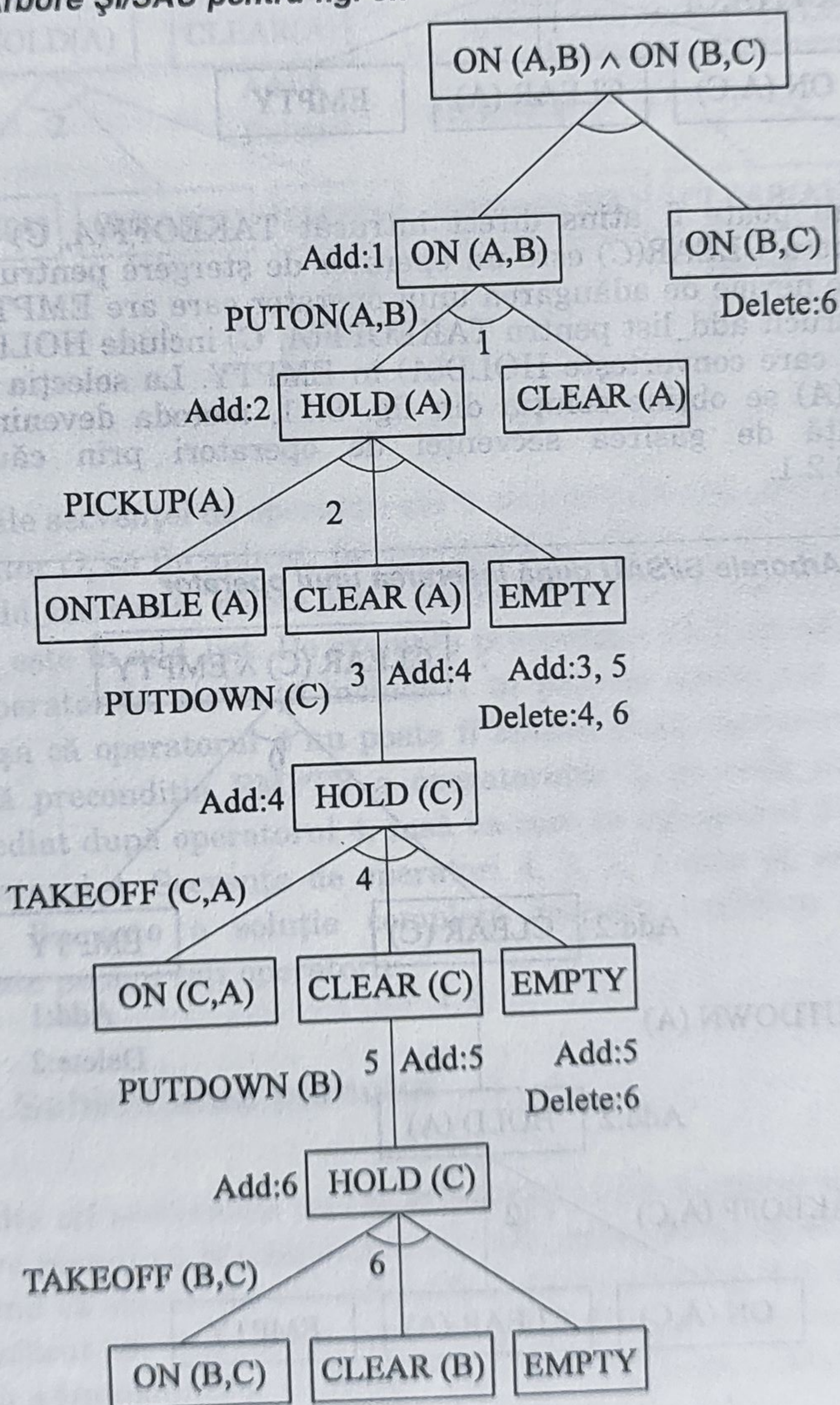
▼ Fig. 8.11. Arborele ȘI/SAU după inserarea unui operator



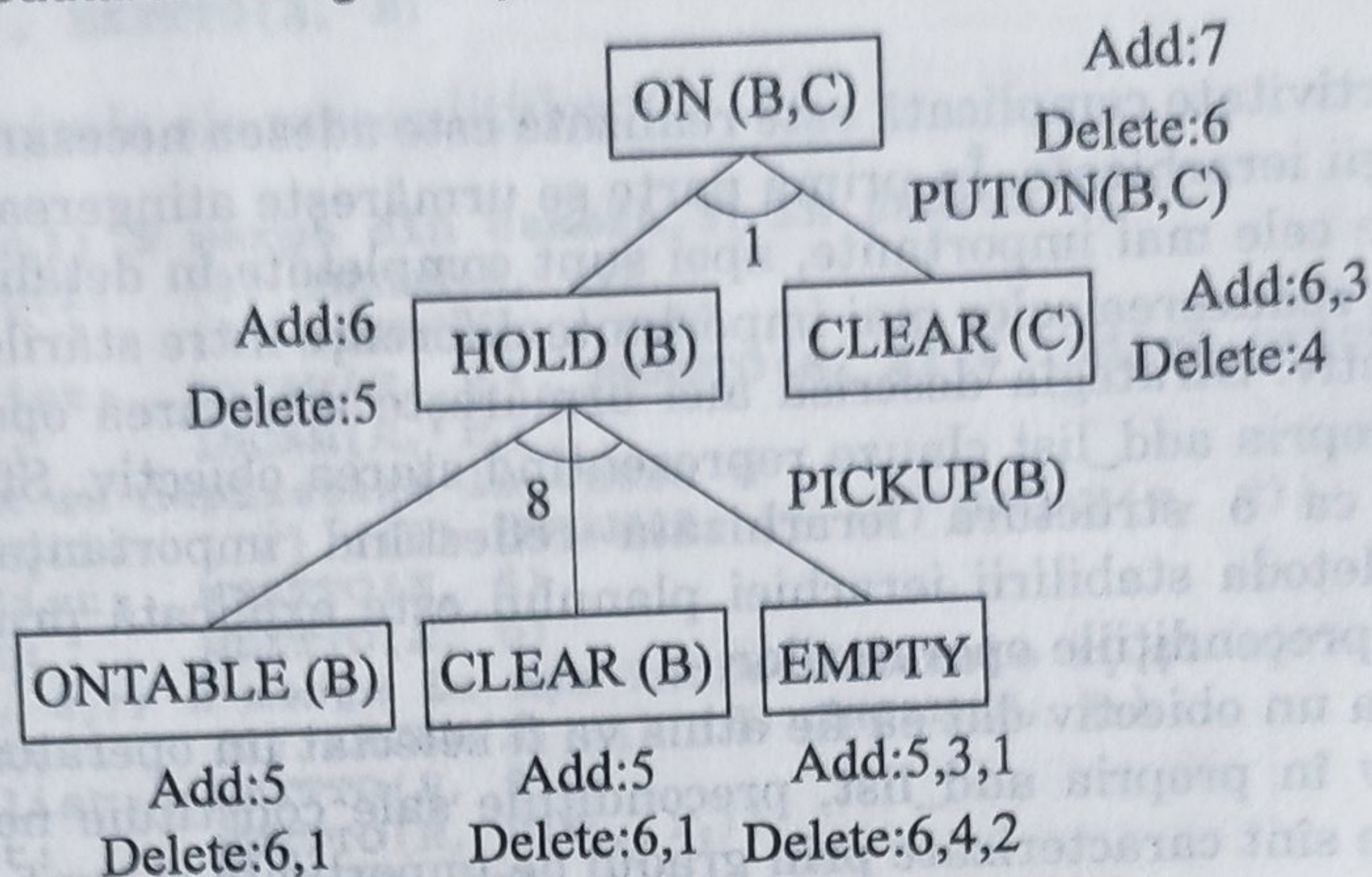
▼ Fig. 8.12. Problema ce necesită revizuirea planului



▼ Fig. 8.13. Arbore ȘI/SAU pentru fig. 8.7



▼ Fig. 8.14. Căutarea adăugând operatori



O serie de probleme nu pot fi rezolvate prin simpla inserare a unui singur operator. În acest caz este necesară construirea unui întreg plan, fără contradicții, prin urmărirea obiectivului, având un operator final de ștergere și asigurând adăugarea succesivă a operatorilor care au acest obiectiv în propria *add_list*.

Se consideră problema din fig. 8.12. Un plan obținut prin separarea obiectivului și simpla inserare a operatorilor este arătat în fig. 8.13. Se poate observa că $ON(B, C)$ a fost distrus pentru a atinge $ON(A, B)$. De aceea se adaugă operatorul $PUTON(B, C)$ care are $ON(B, C)$ în propria sa *add_list*. Așa cum se observă din fig. 8.14, propriile precondiții $HOLD(B)$ și $CLEAR(C)$ devin noi obiective. $HOLD(B)$ nu satisface condițiile inițiale și nu apar operatori adăugați după ștergerea lui. Așa că nu se poate combina imediat ca în fig. 8.13. Din acest motiv se adaugă un operator care conține $HOLD(B)$ în propria *add_list*. Dacă se selectează $PICKUP(B)$ adăugarea și ștergerea operatorilor cu precondițiile lor este arătată în fig. 8.14.

Urmărind fig. 8.14 se observă că după aplicarea operatorului 3 toate precondițiile lui $HOLD(B)$ și $CLEAR(C)$ sînt satisfăcute. De aceea la acest moment se pot găsi operatori de adăugare și ștergere pentru fiecare din obiectivele fig. 8.14 prin recalcularea și verificarea secvenței de operatori din fig. 8.13 ce pot fi inserați între 2 și 3. În acest caz secvența poate fi inserată, și un plan în care aceasta nu este în contradicție cu oricare din operatori (secvența 6, 5, 4, 3, 8, 7, 2, 1) este dat. În general, se poate afirma în concluzie ca planul cere viitoare revizui, operatorii nenecesari vor fi șterși.

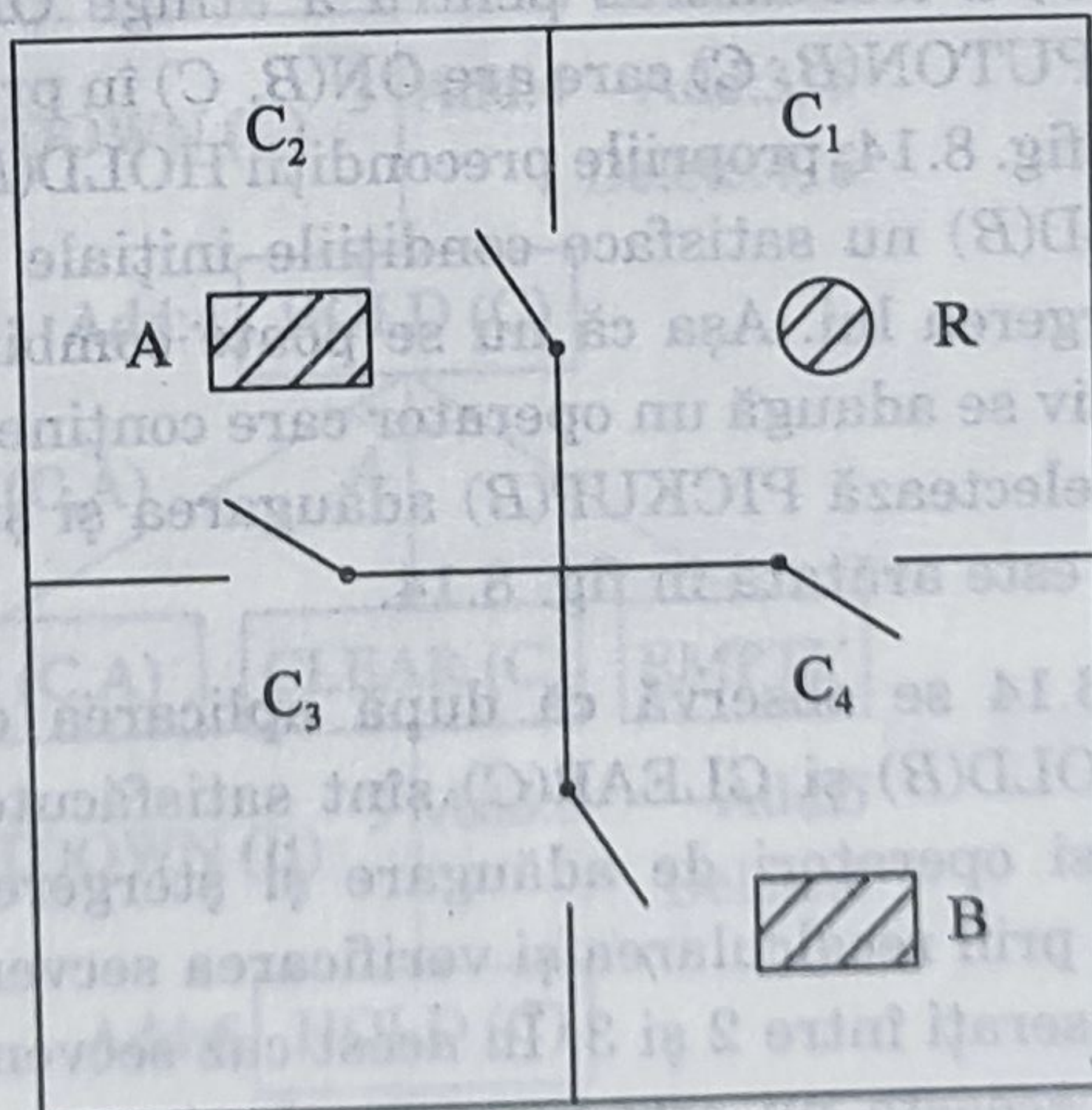
8.2.5. Planuri ierarhizate

Cînd o activitate complicată este realizată este adesea necesară elaborarea unei strategii ierarhizate. În prima parte se urmărește atingerea acelor părți ce par să fie cele mai importante, apoi sunt completate în detaliu strategiile GPS pentru reducerea celor mai importante diferențe între stările specifice și starea obiectiv. Strategia descrisă aici urmărește selectarea operatorilor ce conțin în propria *add_list* clauze reprezentînd starea obiectiv. Strategia este prezentată ca o structură ierarhizată reflectînd importanța diferitelor obiective. Metoda stabilirii ierarhiei planului este explicată prin asignarea ierarhiei la condițiile operatorilor.

Pentru ca un obiectiv dat să fie atins va fi selectat un operator ce conține acel obiectiv în propria *add_list*, condițiile sale constituie noi obiective. Condițiile sînt caracterizate prin gradul de importanță și sunt considerate în funcție de acesta. În acest mod se aranjează obiectivele într-o ierarhie de importanță. Ele vor fi utilizate în stabilirea planului de atingere a celui mai important obiectiv. Planul va fi permanent rafinat pentru includerea și a celor mai puțin importante obiective.

În problema din fig. 8.15 robotul R este localizat în camera C_1 și trebuie să mute cutiile A și B localizate în camerele C_2 și respectiv C_4 , una după alta în camera C_3 . Între camere există uși ce sunt specificate prin $U(C_i, C_j)$, indicînd o ușa între camerele C_i și C_j . Problema se reprezintă prin:

▼ Fig. 8.15. Exemplu de aplicare a planurilor ierarhizate



Starea inițială:

$INCAM(R, C_1), INCAM(A, C_2), INCAM(B, C_4), NOTOPEN(U(C_1, C_2)),$
 $NOTOPEN(U(C_2, C_3)), NOTOPEN(U(C_3, C_4)), NOTOPEN(U(C_4, C_1))$

Starea obiectiv:

INCAM(A, C₃), NEXTTO(A, B)

Operatori și nivele ale condițiilor:

GOTHRU(r_1, r_2); R merge din camera r_1 în camera r_2
 condiții: (3) INCAM(R, r_1)
 (2) OPEN(U(r_1, r_2))
 delete_list: INCAM(R, r_1), NEXTTO(R, \$) \$ indică orice
 add_list: INCAM(R, r_2)
 GOTOB(u); R se deplasează la cutia u
 condiții: (r) {(3) INCAM(R, r) (3) INCAM(u, r)}
 delete_list: NEXTTO(R, \$)
 add_list: NEXTTO(R, u)
 GOTOU(U(r_1, r_2)); R merge la ușa între camerele r_1 și r_2
 condiții: (3) INCAM(R, r_1) (3) INCAM(R, r_2)
 delete_list: NEXTTO(R, \$)
 add_list: NEXTTO(R, U(r_1, r_2))
 OPEN(U(r_1, r_2)); deschide ușa
 condiții: (1) NEXTTO(R, U(r_1, r_2))
 delete_list: NOTOPEN(U(r_1, r_2))
 add_list: OPEN(U(r_1, r_2))
 PUSHTHRU(u, r_1, r_2); robotul mută cutia u din camera r_1 în camera r_2
 condiții: (3) INCAM(R, r_1)
 (4) INCAM(u, r_1)
 (1) NEXTTO(R, u)
 (2) OPEN(U(r_1, r_2))
 delete_list: INCAM(R, r_1), INCAM(u, r_1), NEXTTO(u, \$)
 add_list: INCAM(R, r_2), INCAM(u, r_2), NEXTTO(R, u)
 PUSHB(u₁, u₂); pune cutia 1 după cutia 2
 condiții: (r) {(4) INCAM(u₁, r) (4) INCAM(u₂, r) (3)
 INCAM(R, r)}
 (1) NEXTTO(R, u₁)
 delete_list: nimic
 add_list: NEXTTO(u₁, u₂)

Axiome

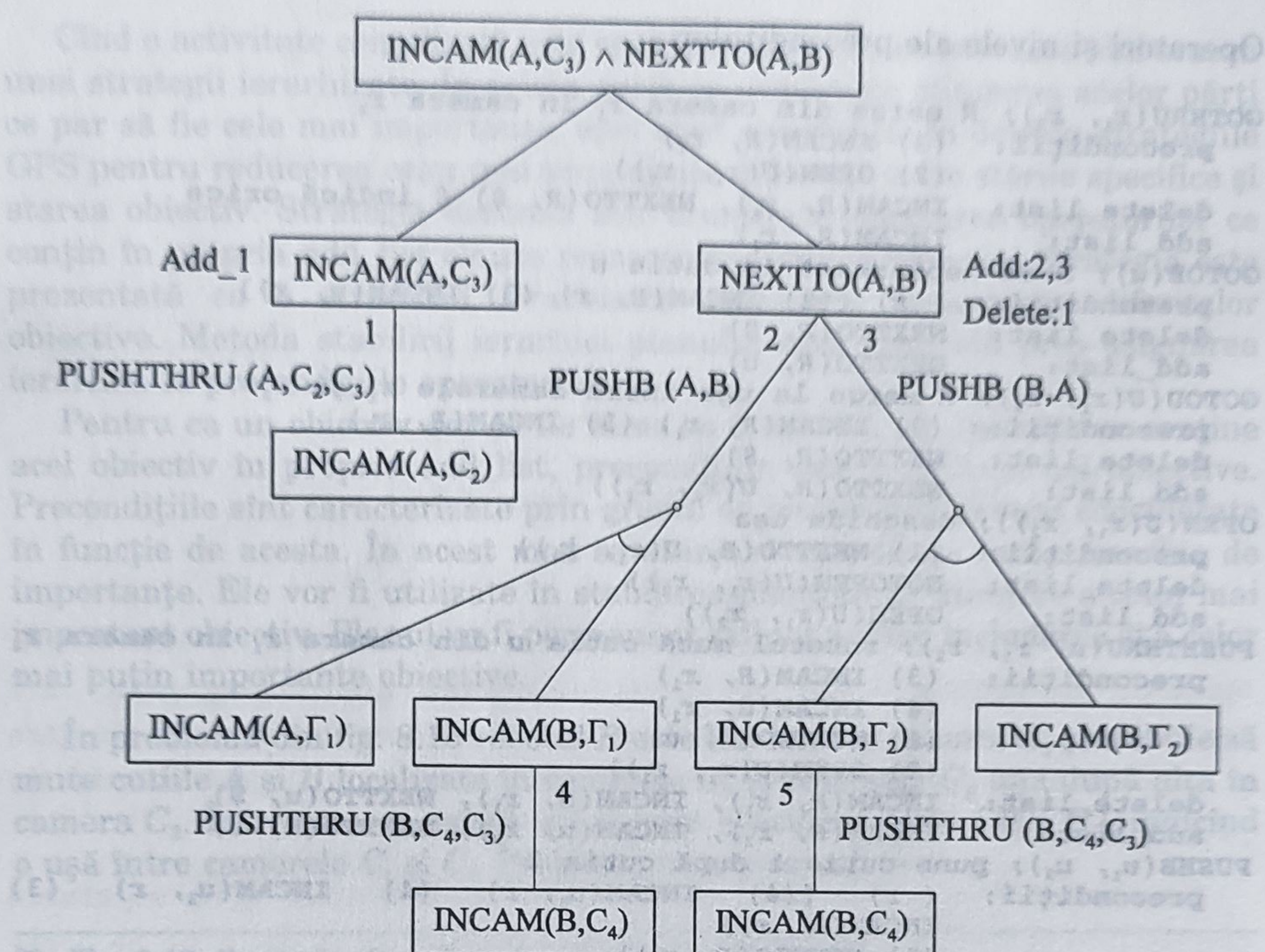
OPEN(U(r_1, r_2)) = OPEN(U(r_2, r_1))
 NEXTTO(u₁, u₂) = NEXTTO(u₂, u₁)

Rezultatele obținute prin prima rezolvare în spațiul abstract al problemei va fi segmentat în mai multe nivele de nivel inferior. Aceste nivele sînt prezentate în continuare:

Nivel 4

Prelucrarea planului care ia în considerație numai condițiile este prezentată în fig. 8.16. Noduri auxiliare sînt inserate în figură întrucît sînt doi operatori care satisfac NEXTTO(A, B). Se poate vedea din add_list și delete_list a operatorului NEXTTO(A, B) că operatorul 1 trebuie aplicat întotdeauna înaintea operatorilor 2 sau 3. De aceea la momentul în care operatorul 2 sau 3 este aplicat INCAM(A, C₃) este true. Din această cauză variabilele r_1 și r_2 ale condițiilor operatorilor 2 și 3 trebuie să fie C₃. Operatorii 4 și 5 sînt de aceea determinați așa cum se vede în figură. La nivelul 4 sînt posibile următoarele două soluții:

▼ Fig. 8.16. Arbore ȘI/SAU în spațiu abstract de nivel 4



$\text{PUSHTHRU}(A, C_2, C_3) \rightarrow \text{PUSHTHRU}(B, C_4, C_3)$
 $\rightarrow \text{PUSHB}(A, B)$
 $\rightarrow \text{PUSHB}(B, A)$

Ambele dintre cele două soluții sînt corecte, însă va deveni clar la nivelul 1 care dintre acestea necesită mai puțini operatori.

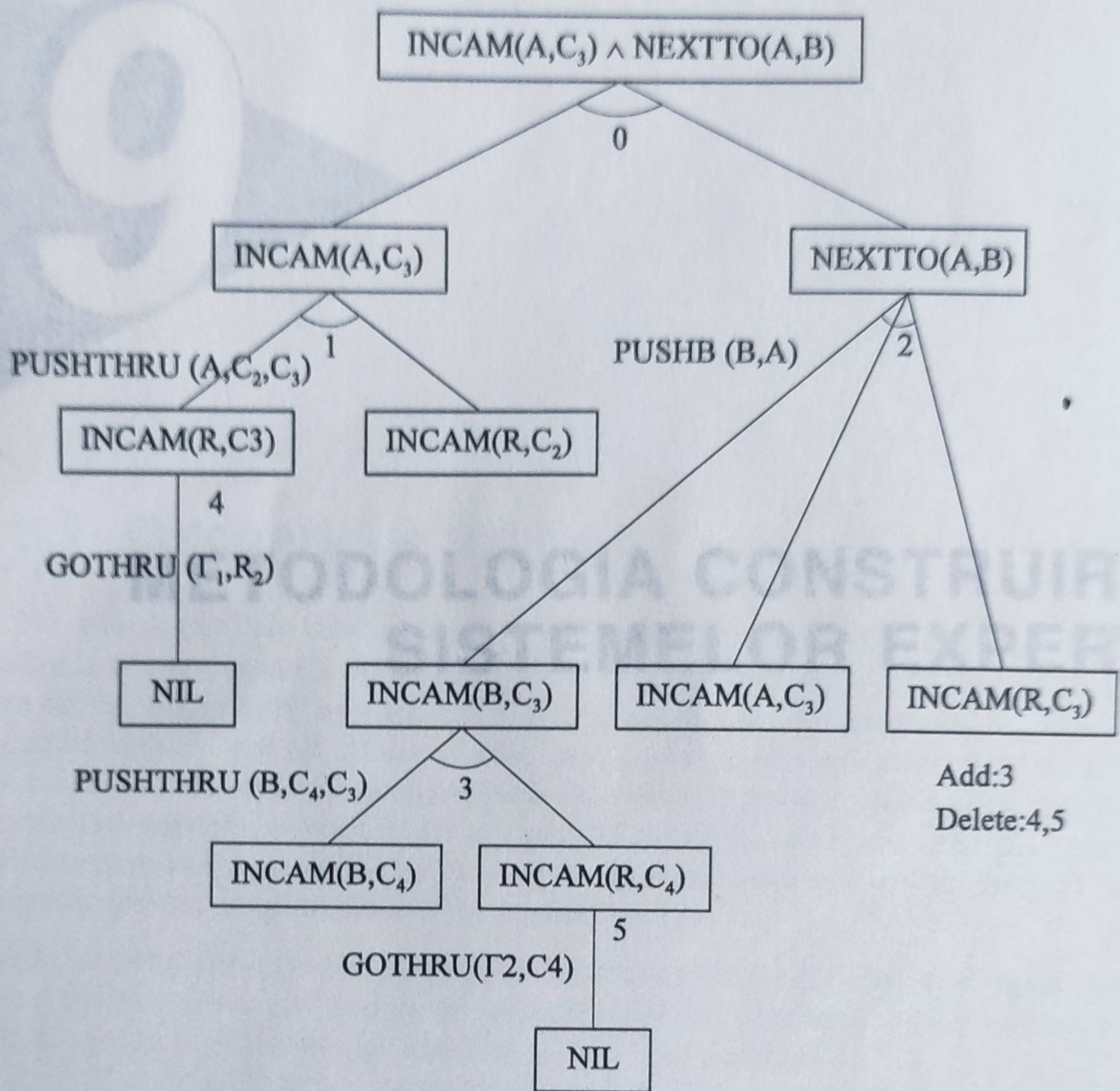
Nivel 3

Dacă se rafinează nivelul 4 prin adăugarea precondițiilor nivelului 4 se obține rezultatul din fig. 8.17. În această figură secvența operatorilor 1, 3, 2 la nivelul 4 este deja dată, așa că timpul de căutare descrește dacă se utilizează ca bază pentru construirea planului. Secvența de operatori a soluției este 4, 1, 5, 3, 2. Aceasta este:

$\text{GOTHRU}(r_1, C_2) \rightarrow \text{PUSHTHRU}(A, C_2, C_3)$
 $\rightarrow \text{GOTHRU}(C_3, C_4) \rightarrow \text{PUSHTHRU}(B, C_4, C_3)$
 $\rightarrow \text{PUSHB}(B, A)$

unde r_2 în fig. 8.17 este C_3 și r_1 este oricare din camerele (fixat la C_1 în planul de nivel 1).

▼ Fig. 8.17. Arbore ȘI/SAU în spațiul abstract de nivel 3



Nivel 2

Cînd se compilează un plan mai detaliat în același mod, următorul plan este obținut (cititorul poate verifica aceasta).

$OPEN(U(r_1, C_2)) \rightarrow GOTHRU(r_1, C_2) \rightarrow OPEN(U(C_2, C_3)) \rightarrow$
 $PUSHTHRU(A, C_2, C_3) \rightarrow OPEN(U(C_3, C_4)) \rightarrow GOTHRU(C_3, C_4) \rightarrow$
 $PUSHTHRU(B, C_4, C_3) \rightarrow PUSHB(B, A)$

Nivel 1

Rezultatul rafinării procesului este:

$OPEN(U(r_1, C_2)) \rightarrow GOTHRU(r_1, C_2) \rightarrow OPEN(U(C_1, C_2)) \rightarrow$
 $GOTO(U(C_2, C_3)) \rightarrow OPEN(U(C_2, C_3)) \rightarrow GOTOB(A) \rightarrow$
 $PUSHTHRU(A, C_2, C_3) \rightarrow GOTOD(U(C_3, C_4)) \rightarrow OPEN(U(C_3, C_4)) \rightarrow$
 $GOTHRU(C_3, C_4) \rightarrow GOTOP(B) \rightarrow PUSHTHRU(B, C_4, C_3) \rightarrow PUSHB(B, A).$

9.1. Categorii de activități

METODOLOGIA CONSTRUIRII SISTEMELOR EXPERT

9.1. Categori de aplicații

Ne propunem să analizăm care sînt diferențele dintre sisteme expert realizate pentru diferite aplicații, dacă acestea se pot grupa în clase, modul în care are loc transferul cunoștințelor. La multe din aceste întrebări nu se poate da un răspuns net și precis, însă, pot apărea elemente ce fac obiectul cercetărilor viitoare. Este necesară în acest context o analiză asupra limitărilor sistemelor expert, cît și a posibilităților viitoare de evoluție. Odată stabilite aceste chestiuni se va încerca să se definească principiile de bază ale metodologiei de construcție a unui sistem expert.

Fără a propune o grupare a sistemelor expert foarte diferite în clase, putem face o diferențiere, în funcție de natura scopului urmărit, deci de utilizare. Astfel, aceste sisteme se pot împărți în trei mari categorii:

- a) SE de clasificare - interpretare, pot realiza spre exemplu clasificarea cauzelor posibile ale unei disfuncționări (cantitative, calitative, vizuale) în scopul de a determina semnificațiile acestor tipuri de date. Acest tip de SE este cel mai vechi (diagnostic, prospecții) și cel mai des utilizat. Dezvoltate cu precădere pentru aplicațiile în medicina cercetărilor au influențat aplicațiile ulterioare rezultînd ceea ce numim diagnostic tehnic. Dintre sistemele utilizate în domeniul medical MYCIN este un exemplu perfect. Sistemul utilizat pentru diagnosticarea infecțiilor bacteriene utilizează o reprezentare a cunoștințelor sub formă de reguli, descriind legăturile între simptome și bolile posibile. Cunoștințele folosite aici sînt cu precădere empirice și de aceea mai sînt numite și "cunoștințe de suprafață". Este util să remarcăm faptul că sistemele ce utilizează cunoștințe de suprafață nu pot să justifice raționamentul ce explică deducțiile realizate. La nivelul raționamentelor utilizate în MYCIN se disting două caracteristici principale:

- raționare înainte (singurul mod).

- cercetare exhaustivă - aplicarea tuturor regulilor posibile pentru a crește sau a diminua factorul de certitudine al concluziei.

Recent, cercetările s-au îndreptat spre alte sisteme capabile să utilizeze "cunoștințe profunde". Dintre sistemele dezvoltate în domeniul diagnozei tehnice amintim DART Stanford University - 82, ACE, Bell - 83, MICRO EXPERT - ISIS, PDS - Carnegie University, PERITUS - Stanford Research Institute, SPEAR - DEC, RECONSIDER - University of California. În acest domeniu construirea unui sistem expert este foarte dificilă din cauză că regulile cu care operează trebuie să fie foarte generale astfel încât să fie asigurată portabilitatea de la o mașină la alta. Pentru diagnoza sistemelor de calcul cu aceeași structura remarcăm faptul că testele legate de memorie, magistrală rămân identice. Cunoștințele pentru astfel de aplicații sunt reprezentate prin descrieri structură-funcție precum și prin relațiile dintre simptome și defecte. Dacă pentru circuitele numerice descrierea relațiilor intrare-ieșire oferea "cunoștințe în profunzime" pentru circuitele complexe acest lucru nu mai este posibil. În această situație se întâlnesc atât "cunoștințe de suprafață" cât și "cunoștințe profunde", însă sistemul expert nu are nevoie de "cunoștințe profunde" complete și foarte detaliate.

b) Sisteme expert de control (monotorizare) - sînt caracterizate prin introducerea noțiunii "timp" care este primordială, și au sarcina de a supraveghea buna evoluție a unui proces (conducere de procese). Aceste sisteme supraveghează evoluția datelor sau semnalelor provenind de la procesul controlat. Ele tratează, deci date continue într-o logică nemonotonă ca în exemplul:

(dacă A este adevărat la un moment T , atunci A este lansat în execuție la momentul $T + Q$).

Pentru aceste sisteme măsura intervalelor regulate de timp trebuie să fie interpretată, generînd declanșarea unei acțiuni imediate, atîta timp cît anumite condiții sînt îndeplinite. Este necesară deci o structură de control ce permite evoluția datelor altfel decît în cazul clasic al diagnosticării. Cunoștințele pot fi reprezentate sub formă de tabele cu date care în funcție de valoarea mărimii controlate selectează strategiile ce vor fi aplicate la momentele următoare. O categorie specială de control este cea destinată optimizării proceselor cooperative.

c) SE de anticipare. Aceste sisteme sînt caracterizate printr-o alocare a resurselor ținînd cont de restricții. Ex. CAO (anticiparea unui rezultat) pentru meteorologie, pentru anticiparea evoluției în viitor pe baza datelor prezente și a evoluțiilor trecute, pentru ordonarea și planificarea producției. Această compartimentare nu a fost făcută de dragul unei clasificări, ea permite studiul și căutarea după caz a sistemului cadru pentru dezvoltarea unei aplicații. Spre exemplu, sistemele expert utilizate în diagnoză nu pot fi utilizate la dezvoltarea aplicațiilor de conducere a proceselor.

În concluzie, este deci indispensabilă determinarea clasei aplicației, și căutarea sistemului de dezvoltare adecvat.

9.2. Alternative în construcția unui sistem expert

Pentru construirea unui sistem sînt posibile două alternative:

- ✕ ▶ existența unei aplicații care în urma analizei justifică abordarea cu un sistem expert în detrimentul programării clasice;
- ▶ existența unui sistem expert cadru pentru o anumită categorie de aplicații la care se atașează o bază de cunoștințe specializată în problema ce se cere rezolvată.

Hotărîrea de a utiliza un sistem expert pentru rezolvarea problemei este luată după examinarea următoarelor situații:

- ✕ ▶ dacă o problemă este bine rezolvată prin modele numerice clasice (programarea liniară sau altele) este inutilă dezvoltarea unui sistem expert. Din contră, dacă problema nu este bine rezolvată în maniera clasică decît în condiții ideale rar atinse, sau numărul parametrilor de natură simbolică puși în joc este dificil de luat în considerație într-un model numeric, atunci se poate apela la o rezolvare utilizînd sisteme expert. Unii autori consideră că sistemele expert sînt lipsite de rigoare în formalism, dar este bine de remarcat faptul că această construcție nu este utilizată decît atunci cînd metodele clasice sînt ineficace. În general, ineficacitatea provine din faptul că problema este dificil formalizată din motive obiective sau subiective;
- ✕ ▶ existența expertului uman, recunoscut ca fiind persoana ce cunoaște bine lucrul studiat. Aceasta explică de ce construcția unui sistem expert depinde de un expert uman pentru baza sa de cunoștințe și de ce dacă problema nu este rezolvată de persoane nu poate fi rezolvată nici de un sistem expert;
- ✕ ▶ dacă evoluția cunoștințelor în domeniu este caracterizată de o dinamică rapidă, necesită frecvențe schimbări, atunci un sistem expert se poate aplica. În această situație se poate folosi și un program clasic, însă amendamentele aparent minore, aduse la acest program îi pot modifica profund comportamentul.

Realizarea programelor clasice este un proces rigid, greu, ce crează dificultăți în elaborarea specificațiilor, proces ce conduce la programe lipsite de suplețe și fără posibilități de îmbunătățire. Din contră, sistemele expert prin însăși construcția lor se pretează la modificări locale numeroase, în condițiile în care modificările nu afectează coerența bazei de cunoștințe:

- ▶ principalele calități ale expertului sînt exprimate în cunoștințele transferate sistemului, cunoștințe ce sînt în general de bun simț, și nu formalizate. Nu trebuie înțeles însă, că sistemul expert nu poate raționa utilizînd cunoștințe puternic formalizate, însă marele câștig rezidă în utilizarea cunoștințelor slab formalizate;

▼ Tabelul 9.1.

Problema Soluția	Precisă și stabilă	Precisă dar evoluează frecvent	Fluctuantă într-un domeniu bine stabilit
Cunoscută	Programare clasică	SE ușor de actualizat ca urmare a evoluției	SE pentru că se poate adapta la fiecare problemă
Necunoscută	SE pentru găsirea soluției, apoi abandonat în favoarea programării clasice	SE pentru căutarea soluției, apoi adaptat pentru exploatare	SE pentru căutarea soluției, ușor de exploatat pentru că se adaptează la problema precisă

- ▶ problemele pentru a căror rezolvare sînt necesare cîteva ore de muncă umană și nu sînt repetitive, nu se pretează la rezolvarea cu sisteme expert pentru a căror rezolvare activitatea este mult mai complexă. Dacă, de exemplu, stabilirea unei configurații pentru un sistem de calcul necesită o zi de muncă umană, ținînd cont de faptul că o problemă similară este rezolvată practic în fiecare zi, un sistem expert este adecvat. Un alt aspect într-o astfel de situație este marcat de imposibilitatea formulării corecte a cererii de către beneficiar, o mare parte din timp fiind ocupată cu discuțiile între specialist și beneficiar. Utilizarea unei interfețe operator "prietenoase" (limbaj natural) a facilitat rezolvarea unor astfel de programe, un exemplu fiind sistemul XCON utilizat la DEC;
- ▶ o cotă a rotației de personal ridicată (cerută des în tehnică) este în aceeași măsură un semn că utilizarea sistemelor expert permite memorarea experienței acumulate de persoanele ce lucrează în domeniu (perenitatea informării);
- ▶ cînd într-o clasă bine definită de probleme, nu se știe care va fi rezolvarea precisă, variabilele de intrare, numărul de terminale, tipul lor, tipuri de comunicație numeroase nu permit scrierea programelor pentru toate variantele în forma clasică. Tabelul 9.1 sumarizează posibilele utilizări ale sistemului expert.

Ne propunem să răspundem la întrebarea: se poate folosi un mecanism de inferență existent, va trebui să-l scriem singuri? A doua cale este mai îmbietoare, autorul este în mijlocul problemei, poate cunoaște toate detaliile programului și este posibil de a adapta structura internă la necesitățile

proprii. Însă trebuie pus în evidență faptul că se realizează un sistem care într-o realizare profesională este necesar să dispună de toate facilitățile pentru dezvoltarea și testarea bazei de cunoștințe.

Pentru a încerca un răspuns la întrebarea: care este sistemul cadru cel mai adecvat, cercetătorii americani decid să facă un test a 8 mecanisme diferite (Emycin, KAS, Expert, OPS5, Rosie, RLL, Age și Hearsay) analizând avantajele și inconvenientele lor. Pentru aceasta, ei cer echipelor specializate pe câte unul din sisteme (echipele ce le-au dezvoltat) să dezvolte în 3 zile un mic sistem expert pentru detectarea, găsirea și informarea persoanelor competente despre o eventuală scăpare într-un sistem de stocare a petrolului sau a altor produse chimice. S-au dedus astfel câteva principii importante:

- ♦ sistemul trebuie să posede numai gradul de generalitate necesar pentru rezolvarea problemei date. Altfel spus, afirmația "cine poate mult poate și puțin" este o eroare, un sistem prea general posedă capacități neutilizate, și aceasta în detrimentul performanțelor;
- ♦ este necesar ca sistemul cadru să aibă o serie de caracteristici, cum sînt:
 - Limbajul de reprezentare a cunoștințelor la nivelul utilizatorului trebuie să fie cît mai simplu și universal;
 - Un mijloc de a ajunge la mecanismul de control dacă generalitatea este mai importantă decît eficacitatea;
 - Capacități de dialog elaborat (limbaj cvasinatural - dacă timpul de elaborare nu este critic);
- ♦ în măsura posibilului se vor utiliza sisteme cadru care au servit la aplicații similare.

Aceste criterii sînt mai mult empirice decît metodologii ale unei teorii precise, însă sînt impuse de faptul că cercetările în domeniu sînt încă incipiente. Pe de altă parte, puține sînt sistemele cadru ce au fost utilizate la suficiente sisteme expert pentru a le testa limitele, eficacitatea și adecvanța. Așa cum s-a încercat să se arate pe parcursul paragrafului prezent, proiectantul are de ales între două alternative:

- a) utilizarea unui sistem cadru existent cu avantajul existenței mecanismului de inferență și a subsistemului cognitiv, dar cu dezavantajul rezultat din necunoașterea intimității sistemului. Mai sus s-a încercat sistematizarea unor criterii pentru alegerea unui astfel de sistem cadru după domeniul aplicației;
- b) construirea completă a sistemului expert.

Este ușor de remarcat că un sistem cadru ce conferă o mare generalitate (mai ales în interfața cu utilizatorul) asigură o realizare mai rapidă. Problema esențială care depinde în speță de calitatea alegerii rămîne totuși performanță, nu totdeauna satisfăcătoare. Construirea unui sistem dedicat asigură totuși mari avantaje, marcate prin suplețe și performanțe deosebite, însă necesită un buget de timp pentru elaborare mult mai ridicat. Sistemul astfel construit va putea fi complet validat numai după o lungă perioadă de

exploatare. Odată găsite mecanismele de construcție, principala activitate constă în dezvoltarea bazei de cunoștințe prin transferul experienței de la om la mașină.

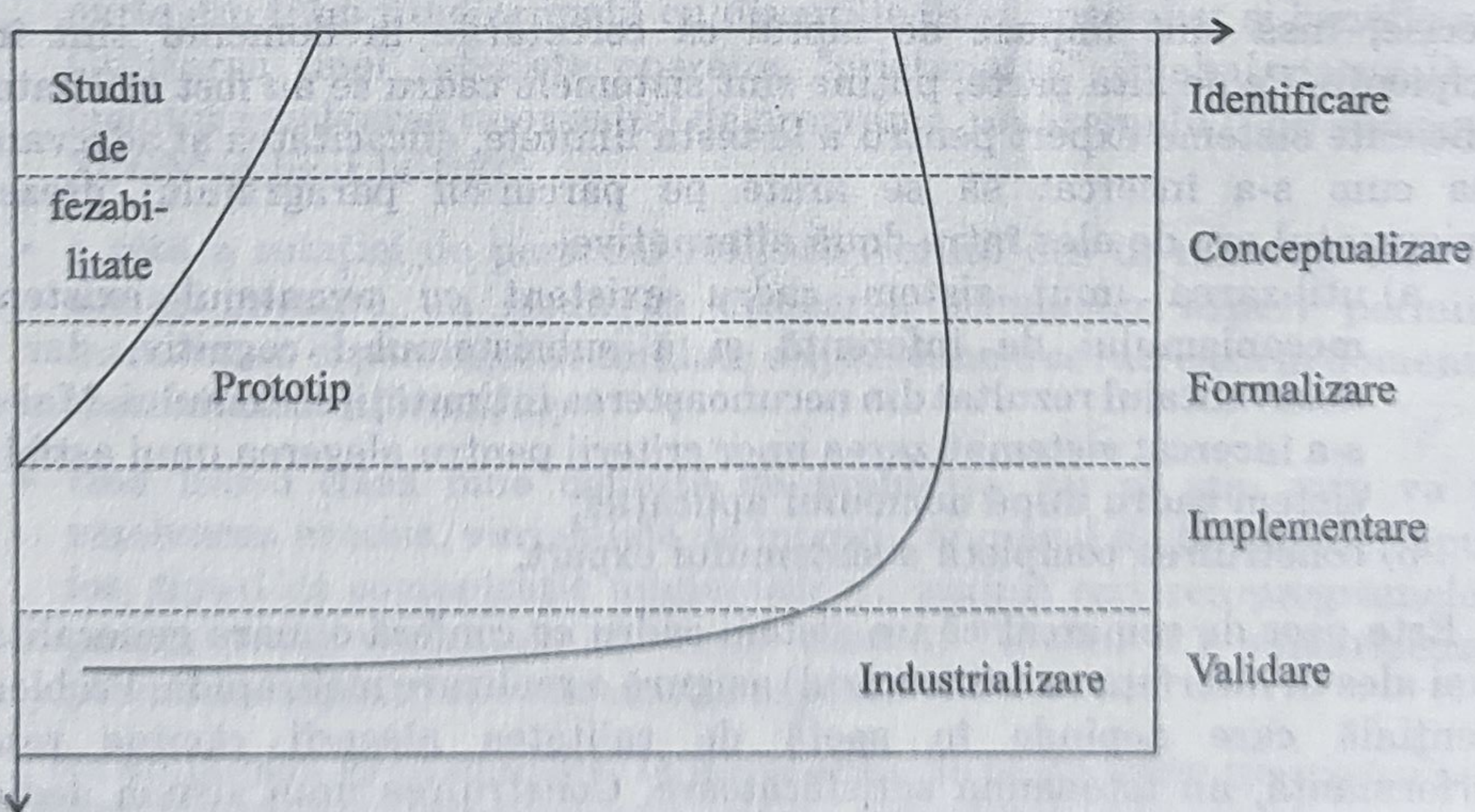
9.3. Etape ale realizării unui sistem expert

Paralel cu derularea procesului de transfer al experienței procesul de realizare a unui sistem expert poate fi descompus în trei perioade:

- A) Un studiu de fezabilitate realizat mai întâi cu ajutorul unei machete de sistem expert;
- B) Realizarea prototipului - vizează construirea unui prototip pe care se experimentează diferitele moduri de reprezentare a cunoștințelor, mecanismele de inferență realizate;
- C) În final, sistemul expert este realizat pornind de la prototip. Dacă rezultatele sînt satisfăcătoare și structura sistemului realizat ca prototip, fie pornind de la zero, permite obținerea eficacității maxime, urmează etapa de industrializare.

În fig. 9.2 se reprezintă cele cinci faze aferente transferului de cunoștințe în conjuncție cu cele trei etape de dezvoltare. Aceasta arată că studiul de fezabilitate este foarte restrîns, însă dezvoltarea prototipului și experimentarea reprezintă părți importante ale procesului de construcție. Diferența între machetă și prototip nu poate fi foarte clară, însă, precis prin prototip se înțeleg subansambluri ale sistemului expert final pe cînd macheta este total independentă.

▼ Fig. 9.2. Etapele dezvoltării sistemului expert



Etapa de prototip permite verificarea rapidă (două sau patru luni) dacă pentru problema dată se justifică utilizarea sistemului expert. Ea presupune realizarea unei machete care nu rezolvă decât câteva cazuri tipice. Macheta este realizată fără a urmări aspecte legate de optimizarea timpului de răspuns și a memoriei ocupate, sau de validarea formalismului utilizat. Inginerul de cunoștințe utilizează pentru aceste reprezentări simple și flexibile, nuclee existente fără să urmărească validarea formalismului utilizat.

La finalul acestui studiu, problema trebuie să fie clar definită, permițând să se ia o decizie asupra desfășurării ulterioare a lucrării. Soluții distincte ale problemei trebuiesc testate și comparate pe parcursul dezvoltării prototipului (care poate fi o extensie a machetei). Pentru aceasta, cel mai comod de utilizat sînt sistemele expert cadru. Acest prototip trebuie, în majoritatea cazurilor să rămînă modest în principiu, el va fi abandonat atunci cînd inginerul de cunoștințe are o idee precisă asupra dezvoltărilor viitoare ale sistemului expert pe baza experiențelor realizate. În final, este preferabilă abandonarea prototipului, modificările nu pot garanta eficacitatea sistemului, chiar dacă prototipul este la un moment dat mult mai bine cunoscut.

Pentru ilustrare să considerăm o comparație imaginară: extensia unui prototip (sau a unei machete) este similară cu aceea de a mări un cart pentru a obține o mașină de formula I. Aceasta nu va atinge niciodată performanțele unei mașini construite în acest scop, deficiențele provin din însăși arhitectura cartului. Diferența esențială între această etapă și etapele precedente rezidă în faptul că la acest nivel se știe foarte exact ce facem și cum facem. Se procedează acum la o rescriere a prototipului testat în etapa anterioară urmărind satisfacerea condițiilor de eficacitate. Baza de cunoștințe continuă să evolueze pe parcurul acestei faze încorporînd cazuri reale și din ce în ce mai complexe (pentru aceasta este necesară dispunerea de un modul de explicare la acest nivel). Sistemul expert începe să devină utilizabil înainte ca baza de cunoștințe și obiectivul fixat inițial să fie realizat complet.

Dezvoltarea în varianta clasică descompune problema în subprobleme realizate independent unele de altele. Din contră, abordarea de viitor este abordarea experimentală și incrementală a machetei, prototipului, sau a tuturor elementelor. Se observă în varianta clasică descompunerea în subprobleme cu rezolvarea acestora în paralel. Astfel rezolvarea este limitată, subproblemele fiind parcurse independent. Construcția printr-un proces experimental și incremental este bazată pe luarea în considerație la anumite etape a tuturor subproblemelor într-o manieră detaliată.

Multe companii se găsesc astăzi la sfîrșitul etapei A, o machetă a fost realizată, însă problemele nu au fost clar delimitate. Dacă decizia de trecere la etapa B era luată (și multe sînt deja luate) partea cea mai importantă începe aici. Ca efect, un studiu asupra domeniului va fi necesar și indispensabil. El permite determinarea cunoștințelor utilizate de expertul uman și maniera de a le reprezenta, sprijinindu-se pe experiența deja

acumulată în acest domeniu (faza de conceptualizare, formalizare). Această etapă este importantă căci ea condiționează urmarea studiului. Lucrurile care sînt efectuate (reprezentarea cunoștințelor, utilitare, mecanism de inferență) determină caracteristicile sistemelor expert viitoare și performanțele lor. Primele rezultate ale studiului pe teren, permit enunțarea criteriilor de căutare a utilităților utilizate în dezvoltarea sistemului expert.

Prototipul obținut, deci baza de cunoștințe nu trebuie să fie voluminoasă, ea servește doar ca suport de decizie pentru autorizarea trecerii la faza următoare. Etapa C începe prin determinarea utilităților corespunzătoare realizării finale (ce pot fi aceleași cu cele ale fazei B). Realizarea acestei etape se sprijină pe prototipul construit în etapa precedentă insistînd pe eficacitatea performanțelor. Baza de cunoștințe a SE final se mărește lent pe măsura avansării proiectului, prin noi cunoștințe și prin rezultatele testelor ce demonstrează validitatea coerenței cunoștințelor. Trebuie amintit că faza de extracție a cunoștințelor este lungă și delicată, ea consumă o mare parte din timpul total de realizare a sistemului expert.

10

APLICAȚII

10.1. Sistem expert pentru supravegherea evoluției mărimilor controlate

Supravegherea buclilor de reglare automate conduse de sisteme industriale este o problemă delicată cu atât mai dificilă cu cât complexitatea procesului este mai mare. Analizând evoluția mărimilor controlate se poate observa că în multe cazuri funcționarea este apropiată de regimul de proiectare, alarmare, atingeri limitelor tehnologice. Pentru înțelegerea acestor fenomene este necesar să se cunoască evoluția mărimilor controlate în timp și să se poată interveni în timp util pentru a evita apariția unor situații periculoase. Este necesar ca sistemul de supraveghere să fie capabil să detecteze și să alerteze operatorul în cazul în care apar anomalii. Acest lucru se poate realiza prin utilizarea unui sistem expert care să analizeze evoluția mărimilor controlate și să detecteze anomalii. Acest sistem poate fi construit pe baza unor metode de inteligență artificială, precum rețele neuronale sau algoritmi de învățare automată. Astfel, sistemul poate să se adapteze la schimbările din proces și să detecteze anomalii înainte ca acestea să devină critice. Acest lucru este foarte important pentru asigurarea siguranței și a calității procesului industrial.

Un alt aspect important al sistemelor de supraveghere este capacitatea lor de a detecta și avertiza operatorul în cazul în care apar anomalii. Acest lucru se poate realiza prin utilizarea unor metode de inteligență artificială, precum rețele neuronale sau algoritmi de învățare automată. Astfel, sistemul poate să se adapteze la schimbările din proces și să detecteze anomalii înainte ca acestea să devină critice. Acest lucru este foarte important pentru asigurarea siguranței și a calității procesului industrial.

10.1. Sistem expert pentru supravegherea evoluției mărimilor controlate

Supravegherea buclelor de reglare aferente conducerii oricărei instalații industriale este o problemă delicată cu atât mai dificilă cu cât complexitatea procesului este mai mare. Analizînd evoluția mărimilor controlate se poate observa că în multe cazuri funcționarea este apropiată regimurilor de prealarmare, alarmare, atingerii limitelor tehnologice. Evoluția în astfel de zone determină costuri de producție mari cu efecte nedorite. Este necesar ca în aceste situații operatorul să fie avizat și să poată interveni în sensul corectării. Apare din aceste considerente ca necesară utilizarea unui sistem expert ce oferă avantaje în adaptabilitatea la situații concrete cît și înglobarea cunoștințelor tipice izvorîte ca urmare a experienței de exploatare pentru fiecare caz în parte. De asemenea, cu cît complexitatea procesului crește și numărul parametrilor supravegheați este mai mare posibilitățile de urmărire a evoluției acestora prin simpla inspectare a istoriei este mai greoaie. Apare deci firească opțiunea de dezvoltare a unui pachet de programe expert pentru supravegherea evoluției.

Se poate considera că în aplicațiile de conducerea proceselor de complexitate mică și medie o structură modulară formată din pînă la 32 de controllere locale ce pot comunica cu un nivel ierarhic superior, nivel ce preia funcțiile specifice consolei operatorului de proces, este adecvată. Prin intermediul fiecăruia dintre controllere pot fi implementați de la simplii algoritmi P, PI, PD, PID, la algoritmi adaptivi și optimali. Oricare ar fi soluția aleasă la nivelul controllerelor locale SE trebuie să răspundă aceluiași deziderate.

10.1.1. Structura sistemului

Comunicația între echipamentul destinat consolei operatorului de proces și controllerele locale asigură actualizarea permanentă a bazei de date asociată fiecărei bucle. Se consideră că baza de date constituită la nivelul consolei operator are o structură modulară și conține toate informațiile de interes asociate buclelor locale. Primii 256 octeti sînt asociați identificatorilor de instalație, bucle, mărimi. În continuare într-o structură identică pentru fiecare controller se constituie o bază de date, fiecărui controller fiindu-i alocate blocuri de 256 octeti. Deplasamentul în blocuri de 256 octeti în baza de date destinată stocării mărimilor de interes asociate fiecărei bucle este dată de adresa de identificare a controllerului pe linia serială. În situațiile în care o structură de acest tip nu permite rezolvarea unei soluții de automatizare complexă mai multe structuri modulare pot fi interconectate la nivelul consolei operator cu un nivel ierarhic superior.

Structura tipică a oricărui sistem expert presupune existența a două mecanisme esențiale: baza de cunoștințe (parte componentă a sistemului cognitiv) și mecanismul de inferență (parte a sistemului rezolutiv). Din motive ușor de înțeles legate de execuția on-line a părții rezolutive, baza de cunoștințe este creată off-line și apoi transportată pe mașina țintă în forma în care este utilizată de către mecanismul inferențial.

10.1.2. Reprezentarea cunoștințelor

Pentru crearea bazei de cunoștințe a fost realizat un editor cu facilități grafice, editor ce crează structura de reguli sub forma de flat-file similar cu informațiile stocate în fișiere de date utilizate în mod curent cu programele de baze de date.

Structura clausală implementată este de tip clausa Horn de forma:

$A_1 \text{ and } A_2 \text{ and } \dots \text{ and } A_n \vdash B$ în care:

- ♦ termenii A_1, A_2, \dots, A_n formează premisa regulii constituită în partea IF;
- ♦ termenul B formează concluzia constituind partea THEN.

Pentru aplicația prezentată un termen din premisa poate avea una din structurile:

nume_param_1 | op_mat | valoare (1)

nume_param_2 | op_mat | val_procentuală*nume_param_2 (2)

în funcție de specificul operației realizate.

Pentru a asigura o uniformitate a păstrării informației ce formează baza de cunoștințe informațiile introduse sînt păstrate într-un fișier cu următoarea structură:

RULE (nr_reg, tip, nume_param_1, op_mat, conect_1, valoare, identif_operator, nume_param_2, conect_2)

în care:

nr_reg este de tip numeric cu trei digiți, câmp ce asigură memorarea numărului de regulă între 1 și 999;

tip câmp de tip caracter (1 octet) ce conține fie caracterul I ce indică primul termen din premisă, fie caracterul A pentru a indica faptul că termenul din premisă este legat de cel anterior prin conectiva logică .and..

nume_param_1 de tip șir caractere ce conține numele simbolic al primului parametru utilizat în operație;

op_mat câmp de tip caracter ce conține unul dintre caracterele <, <=, >, >=, = ce indică tipul operației de comparație;

conect_1 de tip logic în care "true" indică operația de comparație imediată și "false" indică o comparație după înmulțimea parametrului 2 cu un factor;

valoare reprezintă un număr cu valoarea 1 în cazul operațiilor de comparație imediată sau între 1 și 99, când se calculează procentual din al doilea parametru;

identificator_operator ce are asociat "blanc" în cazul comparației imediate sau simbolul % pentru a indica valoarea procentuală;

nume_param_2 de tip șir caractere ce conține numele simbolic al celui de al doilea parametru utilizat în operație;

conect_2 de tip logic cu valorile "true" dacă urmează o nouă condiție în premisă sau "false" dacă este ultima condiție din premisa regulii.

Structura uniformă prezentată mai sus, similară cu cea a fișierelor asociate unei baze de date asigură ușurința vizualizării și modificării bazei de cunoștințe utilizând un program de baze de date obișnuit. În timpul creării bazei de cunoștințe sînt asigurate facilități de ștergere și corecare a regulilor deja introduse. Concluziile regulilor sînt prezentate într-o formă similară cu cele ale termenilor premisei, inferențierea poate duce la obținerea de noi valori logice asociate termenilor premisei.

10.1.3. Implementarea sistemului pentru achiziția cunoștințelor

Implementarea sistemului cognitiv ce crează baza de cunoștințe este realizată în C++. Sursa simplificată a programului de creare a bazei de cunoștințe este prezentată mai jos.

```
# include <stdio.h>
# include <string.h>
# include <graphics.h>
# include <conio.h>
# include <ctype.h>
# include <math.h>
# include <stdlib.h>
# include <dir.h>

// definirea structurii datelor intr-o condiție din clauza IF

struct inreg{
    int flag;
    int nrreg;
    char tip;
```



```

char p1[11];
char opm;
char legv;
int val;
char proc;
char str[5];
char p2[11];
char legl;
};

/* O premisă în forma clausală este constituită din pînă la 6 termeni
cu structură similară, definită în structura inreg */
struct inreg*iff,*and1,*and2,*and3;
struct inreg*and4,*and5,*and6;

//declarații de funcții utilizate
void up(int x, int y, char *s);
void down(int x, int y, char *s);
void cit_par(int x, int y,);
void cit(int x, int y, char ss[10]);
void ecran(void);
void regula(void);
void afişare(void);
void load_rec(void);
void del(void);
void incarc(struct inreg*m);
void mr(int y, struct inreg*m);
char c[2], pathd [MAXPATH], patht[MAXPATH];
char tx[40], s[4];
int caled (void);
int calet (void);
int nreg = 0, l = 0, tip; /*0: numeric, 1: param.*/
FILE *t,*dbf;

/* Proceduri ce determină existența fișierelor ce crează baza de
cunoștințe */
int caled()
{
    int done;
    struct ffbk ffbk;
    strcpy(pathd, "C:\\");
    getcwd(pathd, MAXPATH);
    strcat(pathd, "\\exp.dbf");
    done=findfirst(pathd, &ffbkl, 0);
    return(done);
}

int calet()
{
    int done;
    struct ffbk ffbk;
    strcpy(patht, "C:\\");
    getcwd(patht, MAXPATH);
    strcat(patht, "\\text.txt");
    done=findfirst(patht, &ffbkl, 0);
    return(done);
}

/* Procedura ce realizează încărcarea unei condiții din structura de
tip IF într-un fișier similar cu un fișier de bază de date */
void incarc(struc inreg*m)
{
    int fl, i = 0, j;
    int nr;

```



```

char con;
char pp1[11];
char op;
char llv;
int v;
char pr;
char pp2[11];
char ll;
fl=(unsigned char) m -> flag;
fwrite(&fl, sizeof(fl), 1, dbf);
fprintf(dbf, "%3d", m -> nrreg);
con = m -> tip;
fwrite(&con, sizeof(con), 1, dbf);
strcpy(pp1, m -> p1);
while(pp1p[i] != '\0') i++;
for(j = i; j <= 9; j++)
    pp1[j] = ' ';
pp1[j] = '\0';
fwrite(&pp1, sizeof(pp1), 1, dbf);
op = m -> opm;
fwrite(&op, sizeof(op), 1, dbf);
llv = m -> legv;
fwrite(&llv, sizeof(llv), 1, dbf);
fprintf(dbf, "%5d", m -> val);
pr = m -> proc;
fwrite(&pr, sizeof(pr), 1, dbf);
strcpy(pp2, m -> p2);
i = 0;
while(pp2[i] != '\0') i++;
for(j = i; j <= 9; j++)
    pp2[j] = ' ';
pp2[j] = '\0';
fwrite(&pp2, sizeof(pp2), 1, dbf);
ll = m -> legl;
fwrite(&ll, sizeof(ll), 1, dbf);
}
/* Procedura de formare a regulii în forma clauzală și stocarea
acesteia în scopul generării bazei de cunoștințe */
void load_rec(void)
{
    int i = 0, j;
    incarc(iff);
    if (1==2) incarc(and1);
    if (1==3)
    {
        incarc (and1);
        incarc (and2);
    }
    if (1==4)
    {
        incarc (and1);
        incarc (and2);
        incarc (and3);
    }
    if (1==5)
    {
        incarc (and1);
        incarc (and2);
        incarc (and3);
        incarc (and4);
    }
    if (1==6)
    {
        incarc (and1);
        incarc (and2);
    }
}

```



```

    incarc (and3);
    incarc (and4);
    incarc (and5);
}
if (l==7)
{
    incarc (and1);
    incarc (and2);
    incarc (and3);
    incarc (and4);
    incarc (and5);
    incarc (and6);
}
while (tx[i] != '\0') i++;
for(j = i; j <= 39; j++)
    tx[j] = ' ';
tx[j] = '\0';
fwrite(&tx, sizeof(tx), 1, t);
}
/* Procedura ce realizează interfața cu utilizatorul la crearea bazei
de cunoștințe */
void ecran(void)
{
    char ss[3];
    setcolo(11);
    rectangle(268, 12, 370, 32);
    setcolor(11);
    outtextxy(275, 20, "RULE NO:");
    itoa(nreg, ss, 10);
    outtextxy(344, 20, ss);
    setfillstyle(1, 0);
    bar(0, 179, 640, 480);
    up(10, 180, "If");
    up(10, 220, "Then");
    up(10, 260, "And");
    setfillstyle(1, 11);
    bar(0, 300, 64, 314);
    setcolor(4);
    rectangle(0, 299, 65, 315);
    setcolor(15);
    rectangle(0, 298, 66, 316);
    setcolor(1);
    outtextxy(2, 304, "CHOICE:");
    c[0] = getch();
    c[1] = '\0';
    c[0] = toupper(c[0]);
    while ((c[0] != 73) && (c[0] != 65) && (c[0] != 84)) /* I, A sau
T */
    {
        c[0] = getch();
        c[1] = '\0';
        c[0] = toupper(c[0]);
    }
    outtextxy(57, 304, c);
}
/* Proceduri grafice utilizate de interfața utilizator */
void up(int x, int y, char *s)
{
    setfillstyle(1, 11);
    bar(x + 1, y + 1, x + 39, y + 14);
    setcolor(4);
    rectangle(x, y, x + 40, y + 15);
    setcolor(15);

```



```

line(x + 1, y - 1, x + 41, y - 1);
line(x + 2, y - 2, x + 42, y - 2);
line(x + 3, y - 3, x + 43, y - 3);
line(x + 41, y - 1, x + 41, y + 14);
line(x + 42, y - 2, x + 42, y + 13);
line(x + 43, y - 3, x + 43, y + 12);
setcolor(1);
if (s[0]=='I')
    outtextxy(x + 14, y + 4, s);
if (s[0]=='A')
    outtextxy(x + 8, y + 4, s);
if((s[0] != 'I') && (s[0] != 'A')) outtextxy(x + 4, y + 4, s);
}

void down(int x, int y, char *s)
{
    setfillstyle(1, 0);
    bar(x, y - 3, x + 43, y + 14);
    setfillstyle(1, 11);
    bar(x + 1, y + 1, x + 39, y + 14);
    setcolor(4);
    rectangle(x, y, x + 40, y + 15);
    setcolor(15);
    rectangle(x - 1, y - 1, x + 41, y + 16);
    setcolor(1);
    if (s[0]=='I')
        outtextxy(x + 14, y + 4, s);
    if (s[0]=='A')
        outtextxy(x + 8, y + 4, s);
    if((s[0] != 'I') && (s[0] != 'A')) outtextxy(x + 4, y + 4, s);
}

void cit_par(int x, int y)
{
    setfillstyle(1, 11);
    bar(x + 100, y + 15, x + 279, y + 85);
    setcolor(4);
    rectangle(x + 99, y + 14, x + 280, y + 85);
    setcolor(15);
    line(x + 102, y + 13, x + 281, y + 13);
    line(x + 281, y + 13, x + 281, y + 83);
    setcolor(1);
    outtextxy(x + 102, y + 18, "param. 1:");
    outtextxy(x + 102, y + 33, "operator:");
    outtextxy(x + 102, y + 48, "param. numeric? [y/n]:");
}

void cit(intx, int y, char ss[10])
{
    int i = 0;
    char c1;
    setfillstyle(1, 11);
    bar(x, y + 5, x + 12, y + 8);
    setcolor(1);
    c1 = getch();
    while(c1 != 13)
    {
        if(i < 1) i = 0;
        if(c1==8)
        {
            ss[--i] = '\0';
            bar(x, y, x + 30, y + 8);
        }
        else{

```



```

        ss[i] = c1;
        i++;
        ss[i] = '\0';
    }
    outtextxy(x, y, ss);
    c1 = getch();
}

void mr(int y, struct inreg*m)
{
    int pozx, pozy, k = 0, i = 0;
    char str[10], cc1[2], cc2[2];
    m -> flag = 0x20;
    m -> nreg = nreg;
    if (c[0]==73) /*I*/
    {
        m -> tip = 'i';
        cit_par(30, 180);
        pozx = 30;
        pozy = 180;
    }
    if (c[0]==84) /*T*/
    {
        cit_par(30, 220);
        pozx = 30;
        pozy = 220;
    }
    if (c[0]==65) /*A*/
    {
        m -> tip = 'a';
        cit_par(30, 250);
        pozx = 30;
        pozy = 250;
    }
    setcolor(1);
    line(pozx + 194, pozy + 23, pozx + 202, pozy + 23);
    delay(800);
    cit(pozx + 190, pozy + 18, m -> p1);
    setcolor(1);
    line(pozx + 194, pozy + 38, pozx + 202, pozy + 38);
    delay(800);
    cit(pozx + 190, pozy + 33, cc1);
    m -> opm = cc1[0];
    setcolor(1);
    line(pozx + 268, pozy + 54, pozx + 274, pozy + 54);
    delay(800);
    cc2[0] = getch();
    cc2[1] = '\0';
    while ((cc2[0] != 'y') && (cc2[0] != 'n'))
    {
        cc2[0] = getch();
        cc2[1] = '\0';
    }
    setcolor(11);
    line(pozx + 268, pozy + 54, pozx + 274, pozy + 54);
    setcolor(1);
    outtextxy(pozx + 268, pozy + 48, cc2);
    if (cc2[0]=='y')
    {
        outtextxy(pozx + 102, pozy + 63, "valoare :");
        tip = 0; /* param. numeric */
    }
    if (cc2[0]=='n')
    {
        outtextxy(pozx + 102, pozy + 60, "valoare :");
        outtextxy(pozx + 102, pozy + 73, "param. 2:");
        tip = 1; /* param. nenumeric */
    }
}

```



```

    }
    if (tip==0)
    {
        setcolor(1);
        line(pozx + 190, pozy + 68, pozx + 198, pozy + 68);
        delay(800);
        cit(pozx + 190, pozy + 63, m -> str);
        m -> val = atoi(m -> str);
        m -> p2[0] = '\0';
        m -> legv = 'f';
    }
    if (tip==1)
    {
        setcolor(1);
        line(pozx + 190, pozy + 64, pozx + 198, pozy + 64);
        delay(800);
        cit(pozx + 190, pozy + 59, m -> str);
        while (m -> str[i] != '\0') i++;
        if (m -> str[i - 1] == '%')
        {
            m -> proc = '%';
            m -> str[i - 1] = '\0';
        }
        else m -> proc = ' ';
        m -> val = atoi(m -> str);
        setcolor(1);
        line(pozx + 190, pozy + 78, pozx + 198, pozy + 78);
        delay(800);
        cit(pozx + 190, pozy + 73, m -> p2);
        m -> legv = 't';
    }
    setcolor(11);
    outtextxy(216, 340, "ESC: return to the main menu");
    cc2[0] = getch();
    cc2[1] = '\0';
    while (cc2[0] != 27)
    {
        cc2[0] = getch();
        cc2[1] = '\0';
    }
    setcolor(11);
    rectangle(180, y - 3, 450, y + 12);
    setfillstyle(1, 0);
    bar(181, y - 3, 449, y + 12);
    setcolor(11);
    outtextxy(182, y, s);
    outtextxy(213, y, m -> p1);
    outtextxy(297, y, cc1);
    outtextxy(315, y, m -> str);
    while (m -> str[k] != '\0') k++;
    if (tip==1)
    {
        if (m -> proc == '%') outtextxy(315 + k*8, y, "%");
        else outtextxy(315 + k*8, y, "*");
        outtextxy(315 + (k + 1)*8, y, m -> p2);
    }
    m -> legl = 'f';
}
/* Procedura ce utilizează interfața grafică cu utilizatorul pentru
introducerea regulilor ce formează baza de cunoștințe */
void regula(void)
{
    char cc[2], str[10], ch;
    int y, lung, i = 0;
    nreg++;
    ecra();

```



```

do
{
    switch(c[0])
    {
        case 'I': down(10, 180, "If");
            strcpy(s, "IF");
            iff = (struct inreg*) malloc(sizeof(struct inreg));
            mr(52, iff);
            setcolor(11);
            line(180, 46, 450, 46);
            ecran();
            y = 50;
            break;

        case 'A': down(10, 260, "And");
            strcpy(s, "AND");
            if (1==1)
            {
                iff -> leg1 = 't';
                and1 = (struct inreg*) malloc(sizeof(struct inreg));
                y = 65;
                mr(65, and1);
                ecran();
            }
            if (1==2)
            {
                and1 -> leg1 = 't';
                and2 = (struct inreg*) malloc(sizeof(struct inreg));
                mr(80, and2);
                y = 80;
                ecran();
            }
            if (1==3)
            {
                and2 -> leg1 = 't';
                and3 = (struct inreg*) malloc(sizeof(struct inreg));
                mr(95, and3);
                y = 95;
                ecran();
            }
            if (1==4)
            {
                and3 -> leg1 = 't';
                and4 = (struct inreg*) malloc(sizeof(struct inreg));
                mr(110, and4);
                y = 110;
                ecran();
            }
            if (1==5)
            {
                and4 -> leg1 = 't';
                and5 = (struct inreg*) malloc(sizeof(struct inreg));
                mr(125, and5);
                y = 125;
                ecran();
            }
            if (1==6)
            {
                and5 -> leg1 = 't';
                and6 = (struct inreg*) malloc(sizeof(struct inreg));
                mr(125, and6);
                y = 140;
                ecran();
            }
            break;
    }
    1++;
} while (c[0] != 84); /*'T'*/
setcolor(11);

```



```

line(180, y + 14, y + 14);
strcpy(s, "THEN");
down(10, 220, "Then");
setfillstyle(1, 11);
bar(80, 222, 400, 232);
setcolor(4);
rectangle(79, 221, 401, 233);
setcolor(15);
rectangle(78, 220, 402, 234);
ch = getch();
while(ch != 13)
{
    if(i < 1) i = 0;
    if(ch==8)
    {
        tx[--i] = '\0';
        bar(80 + i*8, 222, 80 + (i + 1)*8, 230);
    }
    else{
        tx[i] = ch;
        i++;
        tx[i] = '\0';
    }
    setcolor(1);
    outtextxy(80, 224, tx);
    ch = getch();
}
setfillstyle(1, 0);
bar(0, y + 15, 640, 480);
lung = 0;
while(tx[lung] != '\0') lung++;
setcolor(11);
rectangle((588-lung*8)/2, y + 25, (666 + lung*8)/2, y + 40);
outtextxy((594-lung*8)/2, y + 29, "Then");
outtextxy((664-lung*8)/2, y + 29, tx);
setcolor(11);
outtextxy(240, 300, "All is correct? [Y/N]");
c[0] = getch();
c[1] = '\0';
c[0] = toupper(c[0]);
while((c[0] != 'Y') && (c[0] != 'N'))
{
    c[0] = getch();
    c[1] = '\0';
    c[0] = toupper(c[0]);
}
if (c[0]==78) /*'N'*/
{
    setfillstyle(1, 0);
    bar(200, 300, 420, 325);
    setcolor(11);
    outtextxy(228, 300, "Delete or Modify the rule?");
    cc[0] = getch();
    cc[1] = '\0';
    while ((cc[0] != 100) && (cc[0] != 109) /* d sau m */)
    {
        cc[0] = getch();
        cc[1] = '\0';
    }
    switch(cc[0])
    {
        case 'd': nreg--;
            1 = 0;
            setfillstyle(1, 0);
            bar(0, 0, 640, 480);
            break;
    }
}

```



```

case 'm': nreg--;
setfillstyle(1, 0);
bar(200, 300, 440, 320);
setcolor(11);
outtextxy(208, 300, "To modify must retype the rule!");
outtextxy(208, 315, "Please star with 'IF'!");
getch();
l = 0;
regula();
break;
}
}
if (c[0]==89) /*'Y'*/
{
setfillstyle(1, 0);
bar(0, 0, 640, 480);
}
}
/* Procedura ce permite inspectarea bazei de cunoștințe create */
void afișare(void)
{
int i, nr, poz = 0, x = 1, y = 1, lg;
char buf[40], s1[10], s[3];
setfillstyle(1, 0);
bar(0, 0, getmaxx(), getmaxy());
printf("\n\n If you want to delete some rules, please");
printf("\n memorate their number! OK?");
getch();
nr = 1;
fseek(dbf, 321, SEEK_SET);
fseek(r, 0, SEEK_SET);
setfillstyle(1, 0);
bar(0, 0, getmaxx(), getmaxy());
setcolor(11);
outtextxy(0, 5, itoa(nr, s1, 10));
lg = strlen(s1);
outtextxy(lg*8, 5, ".");
s1[0] = '\0';
do {
while((poz != 32) || (s1[0] != 'f'))
{ fread(buf, 34, 1, dbf);
s1[0] = buf[4];
s1[1] = '\0';
if(s1[0]=='i')
{ gotoxy(x + 4, y);
printf("If");
}
if(s1[0]=='a')
{ x = 1;
y++;
gotoxy(x + 4, y);
printf("And");
}
for(i = 5; i <= 15; i++)
s1[i - 5] = buf[i];
s1[i - 5] = '\0';
gotoxy(x + 8, y);
printf("%s", s1);
s1[0] = buf[15];
s1[1] = '\0';
gotoxy(x + 18, y);
}
}

```



```

printf("%s", s1);
for(i = 17; i <= 21; i++)
    s1[i - 17] = buf[i];
s1[i - 17] = '\0';
gotoxy(x + 20, y);
printf("%s", s1);
if (buf[16] == 't')
{
    gotoxy(x + 26, y);
    if (buf[22] == '%') printf("%");
    else printf("*");
    for(i = 23; i <= 32; i++)
        s1[i - 23] = buf[i];
    s1[i - 23] = '\0';
    gotoxy(x + 27, y);
    printf("%s", s1);
}
poz = 32;
s1[0] = buf[33];
s1[1] = '\0';
}
poz = 0;
x = 5;
y++;
nr++;
fread(buf, sizeof(buf), 1, t);
buf[40] = '\0';
gotoxy(x, y);
printf("Then %s", buf);
if (y > 23)
{
    printf("Press any key to continue...");
    getch();
    setfillstyle(1, 0);
    bar(0, 0, getmaxx(), getmaxy());
    gotoxy(1, 1);
}
nreg--;
x = 1;
y++;
setcolor(11);
if(nreg != 0)
{
    outtextxy(0, (y - 1)*13, itoa(nr, s2, 10));
    lg = strlen(s2);
    outtextxy(lg*8, (y - 1)*13, ".");
}
} while (nreg != 0);
getch();
}
/* Implementarea mecanismului de ștergere a unor reguli existente */
void del(void)
{
    int n, d[20], i, pozd = 0;
    char buf[33], ss[4];
    int fl, luncg, j = 0;
    char ch;
    setfillstyle(1, 0);
    bar(0, 0, getmaxx(), getmaxy());
    gotoxy(20, 2);
    printf("Delete any rule? [y/n]:");
    scanf("%c", &ch);
    while((ch != 'y') && (ch != 'n')) scanf("%c", &ch);
    if(ch == 'y')
    {
        fl = (char) 0x2A;
    }

```



```

printf("\n Enter numers of rules, in ascendent order.");
printf("\n ESC = finish, BACKSPACE = delete.OK?");
ch = getch();
i = 0;
while(ch != 27)
{
    do
    {
        ss[j] = getch();
        j++;
        if (ss[j - 1] == 8) j--;
    } while(ss[j - 1] != 13);
    ss[j] = '\0';
    printf("\n\r%s", ss);
    n = atoi(ss);
    d[i] = n;
    i++;
    ch = getch();
}
i = 0;
fseek(dbf, 321, SEEK_SET);
lung = strlen(d);
while(lung != 0)
{
    fread(buf, 34, 1, dbf);
    for(n = 0, n <= 2; n++)
        ss[n] = buf[n + 1];
    ss[n] = '\0';
    n = atoi(ss);
    if(n == d[i])
    {
        fseek(dbf, pozd, SEEK_CUR);
        fwrite(&f1, sizeof(f1), 1, dbf);
    }
    i++;
    pozd += 34;
    lung--;
}
}
/* Programul principal pentru generarea regulilor de producție */
void main(void)
{
    char ch;
    int nri = 0;
    int gdr = DETECT, gm;
    initgraph(&gdr, &gm, "C:\\\\TC\\BGI");
    if (caled())
    {
        printf("\n File not found!");
        getch();
        exit(0);
    }
    if((dbf = fopen(pathd, "r + w")) == NULL)
    {
        printf("\nCannot open file for reading/writing.");
        getch();
        exit(0);
    }
    if (calet())
    {
        printf("\n File not found!");
        getch();
    }
}

```



```

    exit(0);
}
if ((t = fopen(patht, "r + w")) == NULL)
{
    printf("\nCan't read from file!");
    getch();
    exit(0);
}
fseek(dbf, 321, SEEK_SET);
fseek(t, 0, SEEK_SET);
ch = 121;
while(ch != 110) /*'N'*/
{
    regula();
    load_rec();
    setcolor(11);
    outtextxy(240, 300, "Add a new rule? [Y/N]");
    ch = getch();
    while((ch != 'y') && (ch != 'n')) ch = getch();
    nri++ = 1;
    l = 0;
}
fseek(dbf, 4, SEEK_SET);
fwrite(&nri, sizeof(nri), 1, dbf);
afisare();
del();
setfillstyle(1, 0);
bar(220, 305, 400, 315);
gotoxy(35, 19);
printf("CIAO!");
ch = getch();
fclose(t);
fclose(dbf);
return 0;
closegraph();
}

```

Se dau mai jos câteva exemple de reguli de producție utilizate:

Regula i: IF: val_intrare > lim_sup_tr

THEN: Traductor defect

Regula j: IF: mod_lucru = AUTO

and: Kp < 1

and: Ti < 1

and: EROARE > 90% ER_MAX

and: TIMP >= 2Ti

THEN: Evoluție incorectă.

10.1.4. Implementarea mecanismului de inferență

Structura clausală Horn asigură minimizarea timpului de execuție asociat mecanismului de inferență. Pentru premisa oricărei reguli, premisa ce conține mai mulți termeni legați prin conectiva logică .and, orice termen a cărei valoare inferențiată este "false" determină asocierea valorii "false" pentru toată premisa, fapt ce duce la următoarea regulă și continuarea procesului inferențial. Rezultatul inferențelor este prezentat operatorului sub forma unui

mesaj în urma căruia operatorul decide dacă rezultatul este efectiv aplicat sau nu. Programul sursă al mecanismului de inferență este prezentat în cele ce urmează:

/* Au fost omise liniile destinate headerelor și declarațiilor.
Operațiile se execută pe baza datelor din baza de date asociată
controllerului în lucru */

```
char pathd[MAXPATH], patht[MAXPATH];
FILE*, dbf, *t;
```

```
void main(void)
```

```
{ char buf[40], s1[10];
```

```
char log, masca;
```

```
int pozd = 0, pozt = 0, flag, i = 0;
```

```
int val, val1, val2, j, nreg = 0;
```

```
int gdr = DETECT, gm;
```

```
initgraph(&gdr, &gm, "C:\\TC\\BGI");
```

```
if (caled())
```

```
{
```

```
printf("\n File not found!");
```

```
getch();
```

```
exit(0);
```

```
}
```

```
if((dbf = fopen(pathd, "r"))==NULL)
```

```
{ printf("\nCannot open file for reading/writing.");
```

```
getch();
```

```
exit(0);
```

```
}
```

```
if (calet())
```

```
{
```

```
printf("\n File not found!");
```

```
getch();
```

```
exit(0);
```

```
}
```

```
if ((t = fopen(patht, "r"))==NULL)
```

```
{ printf("\nCan't read from file!");
```

```
getch();
```

```
exit(0);
```

```
}
```

```
setfillstyle(1, 0);
```

```
bar(0, 0, getmaxx(), getmaxy());
```

```
fseek(dbf, 289, SEEK_SET);
```

```
fseek(t, 0, SEEK_SET);
```

```
while(!eof(dbf))
```

```
{ fread(buf, pozd + 34, 1, dbf);
```

```
s1[0] = buf[0];
```

```
s1[1] = '\0';
```

```
flag = atoi(s1);
```

```
if (flag != 0x2A)
```

```
{ for(i = 1; i <= 3; i++)
```

```
s1[i - 1] = buf[i];
```

```
s1[i - 1] = '\0';
```

```
nreg = atoi(s1);
```

```
for(i = 5; i <= 14, i++)
```

```
s1[i - 5] = buf[i];
```

```
s1[i - 5] = '\0';
```

```
/* caută în baza de date;*/
```

```
/* întoarce valoarea val1.*/
```

```
/* (coresp. lui p1) */
```

```
for(i = 17; i <= 21; i++)
```

```
s1[i - 17] = buf[i];
```



```

s1[i - 17] = '\0';
val = atoi(s1);
if (buf[16]=='t') /* urmează param. p2 */
{
    for(i = 23; i <= 32; i++)
        s1[i - 23] = buf[i];
    s1[i - 23] = '\0';
    /* preia val2 din bd(coresp. lui p2); */
    if (buf[22]=='%') val* = 0.01;
    val* = val2;
}
switch(buf[15])
{
    case '<': if(val1<val) log = 't'; /* compară val cu
            val1.*/
            else log = 'f'; /* întoarce log.*/
            break;
    case '>': if(val1>val) log = 't';
            else log = 'f';
            break;
    case '=': if(val1==val) log = 't';
            else log = 'f';
            break;
}
if (log=='f')
{
    while(val==nreg)
    {
        fread(buf, pozd + 34, 1, dbf);
        for(i = 1; i <= 3; i++)
            s1[i - 1] = buf[i];
        s1[i - 1] = '\0';
        val = atoi(s1);
        pozd+ = 34;
    }
    pozd-=34;
    poz+=40;
}
if ((log=='t') && (buf[33]=='f'))
{
    fread(buf, 40, 1, t);
    i = 0;
    while(buf[i] != '\0') i++;
    setfillstyle(1, 11);
    bar((640 - (i - 1)*10)/2, 100, (640 + (i - 1)*10)/2, 130);
    setcolor(4);
    rectangle((640 - (i - 1)*10)/2 - 1, 99,
        (640 + (i - 1)*10)/2 + 1, 131);
    outtextxy(280, 110, "WARNING!");
    setcolor(15);
    rectangle((640 - (i - 1)*10)/2 - 2, 98,
        (640 + (i - 1)*10)/2 + 2, 132);
    setcolor(1);
    outtextxy((640 - (i - 1)*10)/2, 120, buf);
    exit (1);
}
}
}
pozd+=34

```


10.1.5. Integrarea în aplicație

Întrucît procesul inferențial este singurul ce se execută "online", el este organizat sub forma unui task cu prioritatea cea mai mică față de taskurile ce deserveșc comunicația între consola operator și controllerele conectate serial. Motivul pentru care acest task este privit ca un task de prioritate mică rezidă în faptul că activitatea critică rămîne cea de culegerea datelor. Taskul ce constituie mecanismul de inferență poate furniza rezultate la intervale de timp de aproximativ 10 ori mai mari față de perioada de eșantionare. Pentru ca modulul de inferență să fie instalat ca task este necesară activarea sa ca funcție task într-o manieră similară cu cea descrisă mai jos:

```
void far task_mec_inf(void)
{
    inițializări variabile locale
    while (condiție_task)
    {
        task_mec_inf
    }
}
```

Rezultatele experimentale obținute în mediu simulat (cu baza de date din proces simulată sub forma de fișiere) arată o comportare satisfăcătoare atît din punctul de vedere al timpului de execuție, al comutării corespunzătoare a taskurilor cît și a rezultatelor obținute în urma inferențelor.

10.2. Sistem expert pentru acordarea optimă a buclelor de reglare

Controllerele PID și subseturile acestora P, PI, PD sînt foarte des utilizate în multe aplicații industriale. Găsirea parametrilor de stare constă în mod normal din trei pași:

- ▶ Determinarea aproximativă a parametrilor de acord pornind de la una din formulele standard;
- ▶ Determinarea tipului răspunsului dorit și selectarea unui criteriu ca "quarter-decay" or minimum integrated absolute error (IAE)";
- ▶ Realizarea ajustării necesare a parametrilor pentru obținerea răspunsului dorit;

Operatorul poate evalua performanțele propriiei setări prin modificarea referinței și observarea răspunsului simulînd în acest fel acțiunea perturbațiilor.

Forma canonică a compensatoarelor din această categorie este descrisă prin relația:

$$G_1 = K_p \left(1 + sT_d + \frac{1}{sT_i} \right) \quad (10.1)$$

în care:

K_p - factor de proporționalitate;

T_i - constanta timpului de integrare;

T_d - constanta timpului de derivare;

De cele mai multe ori stabilirea valorilor acestor parametrii, astfel încât performanțele să fie cele dorite este mai mult o artă decît o știință. Gradul de cunoaștere a părții fixate influențează puternic valorile parametrilor de acord. Una din tehnicile de acord bazată pe limita de stabilitate a sistemului este tehnica Ziegler-Nichols. Acordarea empirică după metoda Ziegler-Nichols pornește de la eliminarea efectelor termenului integral și derivativ după care se urmărește creșterea factorului de amplificare pînă cînd se atinge punctul critic de oscilație.

Mărimile factorului de amplificare (K_u) și perioadei de oscilație (P_u) pentru limita de stabilitate vor fi utilizate la calculul parametrilor de acord după relațiile empirice:

$$D_p = 0.6 * K_u \quad T_i = P_u/2 \quad T_d = P_u/8 \quad (10.2)$$

Răspunsul la intrare treapta cu aceste valori de setare obținute rapid, duce la valori ale suprareglajului neacceptabil de mari. O altă tehnică, cea analitică aplicată pentru sisteme caracterizate prin funcția de transfer:

$$G_2(s) = \frac{Ke^{-T_s}}{(1 + sT_1)(1 + sT_2)} \quad (10.3)$$

duce la următorii parametri de acord

$$K_p = 2(T_i + T_2)/3KT, \quad T_i = T_1 + T_2, \quad T_d = T_1T_2/(T_1 + T_2) \quad (10.4)$$

De asemenea, în situația în care procesul controlat este greu de modelat, cum este cazul multor sisteme industriale acordarea este bazată de cele mai multe ori pe intuiție, fără o justificare complet teoretică. În plus, este normal ca operatorii să aibă propriile preferințe pentru cea mai bună traiectorie a răspunsului, fapt ce va determina modificarea parametrilor de acord în sensul obținerii răspunsului dorit. În acest caz nu se mai pune problema definirii matematice a optimalității. Este cunoscut faptul, că mulți dintre operatori consideră ca principale performanțe perioada de oscilație și suprareglajul.

Descrieri ale diferitelor sisteme expert ce realizează ajustarea parametrilor în controllere PID au apărut în literatură. Una din cele mai cunoscute lucrări T.W. Kraus, T.J. Myron - 1984 tratează modul de ajustare prin care se urmărește obținerea celui mai rapid răspuns cu respectarea valorilor maxime permise pentru suprareglaj și dumping. Alte sisteme urmăresc determinarea erorii integrale și calculul factorului de amplificare după identificarea sistemului, alegerea diferiților algoritmi numerici de control bazați pe obiectivele operatorului sau utilizarea regulilor de clasificare în scopul determinării clasei din care face parte instalația. O excelentă lucrare J. Litt -

1990 încearcă să surprindă dependența între performanțe și valorile parametrilor de acord. Abordarea problemei de acordare optimă impune două mari direcții: fundamentarea unei metode de ajustare a parametrilor și integrarea sistemului expert în pachetul de programe cu funcționare în timp real.

10.2.1. Dependente între parametri și performanțe

În vederea utilizării unui sistem expert pentru ajustarea parametrilor o serie de rezultate ale procesului de simulare au fost necesare. În acest sens s-a studiat răspunsul unei bucle de reglare pentru care partea fixată este descrisă prin funcția de transfer:

$$G_2(s) = \frac{e^{-T_s}}{(T_1s + 1)(T_2s + 1)} \quad (10.5)$$

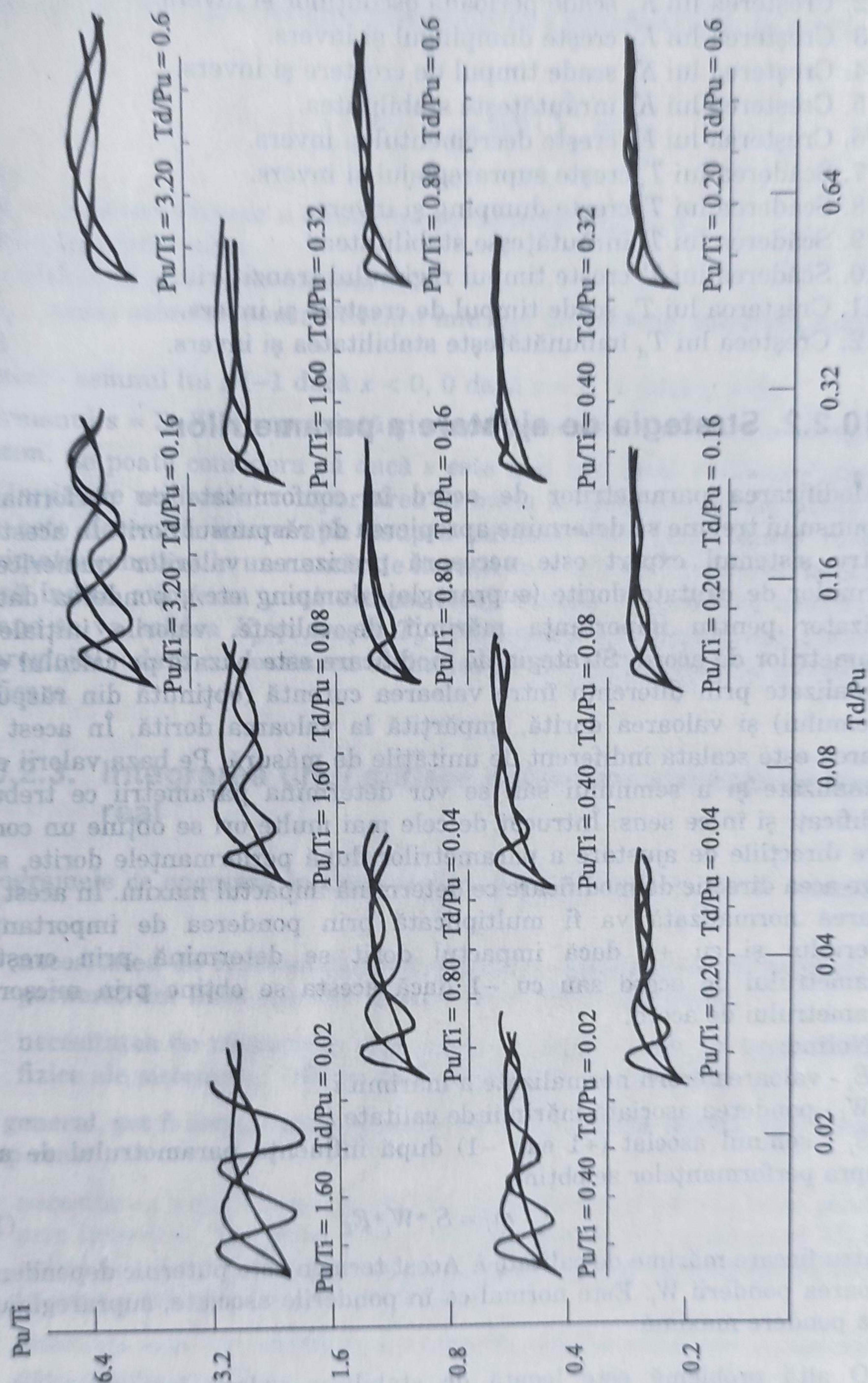
considerînd că acoperă o gamă largă de instalații industriale. În urma studiului răspunsului prin simulare, cât și prin metode teoretice bazate pe caracteristicile de frecvență pot fi obținute dependențele între performanțe și parametri de acord. Pentru a obține dependențele între valorile parametrilor T_v , T_d , K_p și forma traiectoriei răspunsurilor au fost desenate în planul T_d/P_u - P_u/T_v (fig. 10.1). Normalizarea axelor prin P_u ajută la dimensionare și este generală pentru o clasă de sisteme de tipul 10.5. Așa că dacă o instalație poate fi aproximată printr-o funcție de transfer similară cu 10.5 și cu perioada de oscilație determinată experimental, parametri de acord doriți pentru un anumit tip de răspuns pot fi selectați direct din diagramele corespunzătoare. La implementarea prin utilizarea unui sistem expert aceste diagrame sînt utilizate calitativ și nu cantitativ în sensul extragerii dependențelor între performanțe și parametri de acord. Fiecare set de caracteristici este determinat pentru valori diferite ale factorului de amplificare K_p .

Aprecierea adecvenței răspunsului este caracterizată de o serie de mărimi cu semnificația cunoscută în teoria sistemelor.

1. *Suprareglaj* σ_1 .
2. Raportul între valoarea corespunzătoare celui de al doilea și primul maxim numit și *decrement* $\delta = \sigma_3/\sigma_1$.
3. *Timpul de creștere* t_c .
4. *Dumping* $\psi = (\sigma_3 - \sigma_2)/(\sigma_1 - \sigma_2)$ acest factor este mare dacă răspunsul este puternic oscilant și mic dacă răspunsul este caracterizat de oscilații mici.
5. *Perioada oscilațiilor* (P_0).
6. *Timpul regimului tranzitoriu* ce nu este specificat direct, mărimea sa fiind automat cuprinsă în valorile mărimilor δ și ψ .

Din analiza răspunsului pentru diferiți parametri de acord pot fi deduse o serie de reguli privind dependența între mărimile de calitate formulate mai sus și parametri de acord.

▼ Fig. 10.1. Caracterizare răspunsuri



Observațiile calitative sînt:

1. Creșterea lui K_p crește suprareglajul și invers.
2. Creșterea lui K_p scade perioada oscilațiilor și invers.
3. Creșterea lui K_p crește dumpingul și invers.
4. Creșterea lui K_p scade timpul de creștere și invers.
5. Creșterea lui K_p înrăutățește stabilitatea.
6. Creșterea lui K_p crește decrementul și invers.
7. Scăderea lui T_i crește suprareglajul și invers.
8. Scăderea lui T_i crește dumping și invers.
9. Scăderea lui T_i înrăutățește stabilitatea.
10. Scăderea lui T_i crește timpul regimului tranzitoriu.
11. Creșterea lui T_d scade timpul de creștere și invers.
12. Creșterea lui T_d îmbunătățește stabilitatea și invers.

10.2.2. Strategia de ajustare a parametrilor

Modificarea parametrilor de acord în conformitate cu performanțele răspunsului trebuie să determine apropierea de răspunsul dorit. În acest sens pentru sistemul expert este necesară precizarea valorilor numerice ale mărimilor de calitate dorite (suprareglaj, dumping etc.), ponderea dată de utilizator pentru importanța mărimii de calitate, valorile inițiale ale parametrilor de acord. Strategia de modificare este bazată pe calculul erorii normalizate prin diferența între valoarea curentă (obținută din răspunsul sistemului) și valoarea dorită, împărțită la valoarea dorită. În acest mod eroarea este scalată indiferent de unitățile de măsură. Pe baza valorii erorii normalizate și a semnelui său se vor determina parametrii ce trebuiesc modificați și în ce sens. Întrucât de cele mai multe ori se obține un conflict între direcțiile de ajustare a parametrilor după performanțele dorite, se va alege acea direcție de modificare ce determină impactul maxim. În acest scop eroarea normalizată va fi multiplicată prin ponderea de importanță a criteriului și cu +1 dacă impactul dorit se determină prin creșterea parametrului de acord sau cu -1 dacă acesta se obține prin micșorarea parametrului de acord.

Notînd:

E_i - valoarea erorii normalizate a mărimii i

W_i - ponderea asociată mărimii de calitate i

S_i - semnul asociat (+1 sau -1) după influența parametrului de acord asupra performanțelor se obține:

$$m_i = S_i * W_i * E_i \quad (10.6)$$

pentru fiecare mărime de calitate i . Acest termen este puternic dependent de valoarea ponderii W_i . Este normal ca în ponderile asociate, suprareglajul să aibă pondere maximă.

O altă problemă este legată de stabilirea valorii incrementului sau decrementului ce determină noua valoare a parametrului de acord. O strategie

posibilă, utilizată în tehnicile de optimizare, conduce la folosirea unei secvențe Fibonacci $a_k = a_{k-1} + a_{k-2}$ pentru fixarea pasului sau la metode de bisectionare cu șirul $a_k = 2^k$. Se obține pentru noua valoare a parametrului de acord:

$$P(k+1) = P(k) \left[1 + \frac{1}{a_k} \operatorname{sgn} \left(\sum_{i=1}^R m_i \right) \right] \quad (10.7)$$

în care:

$P(k+1)$ - noua valoare a unuia din parametrii: K_p, T_v, T_d ;

$P(k)$ - valoarea veche;

a_k - termenul șirului de ordinul k ;

$\sum m_i$ - suma scorului pentru fiecare mărime de calitate calculată conform (10.6);

$\operatorname{sgn}(x)$ - semnul lui x (-1 dacă $x < 0$, 0 dacă $x = 0$, 1 dacă $x > 0$);

Termenul $s = \sum s_i W_i E_i$ reprezintă și scorul care este o măsură a apropierii de optim. Se poate considera că dacă s este mai mic decât o valoare impusă (dată inițial de utilizator) comportarea în bucla închisă este corespunzătoare și nu este necesară intervenția asupra parametrilor. Dacă scorul rămâne aproximativ constant la un număr de iterații cu o valoare mai mare decât cea impusă indică atingerea unui extrem local. Pentru ca procesul iterativ să continue se va scădea K_p și crește T_v , (parametrii ce influențează puternic suprareglajul) odată cu resetarea ordinului șirului pentru fixarea pasului de modificare.

10.2.3. Integrarea unui sistem expert cu aplicații în timp real

Programele ce operează în timp real au caracteristici ce pot fi rezumate astfel:

- ▶ necesitatea de reactualizare simultană a parametrilor (în paralel), a parametrilor fizici specifici sistemului, cât și a resurselor;
- ▶ necesitatea de răspuns în restricțiile de timp impuse de proprietățile fizice ale sistemului real la care calculatorul este cuplat.

În general, pot fi listate câteva probleme când un sistem expert este utilizat în timp real:

- ▶ necesitatea unui interval mare de timp pentru obținerea unei soluții prin inferență. Din acest motiv o cerință deosebită în realizarea SE cu aplicații în timp real impune noi moduri de reprezentare a cunoștințelor în sensul scăderii timpului de rulare;
- ▶ inferența este întreruptă de evenimente externe asincrone ce necesită prelucrarea unor date;
- ▶ pentru tratarea a doua sau mai multe evenimente cu aceeași prioritate, SE trebuie să fie capabil să execute inferențe în paralel;

- ▶ multe din datele noi, pot fi contradictorii cu cele deja alocate. Din acest motiv este necesar ca baza de cunoștințe să nu aibă o construcție monolitică;
- ▶ valorile fizice continue provenite de la senzori vor fi transformate în simboluri logice cu care se operează.

Ideea de bază a integrării unui SE într-o aplicație în timp real pornește de la organizarea întregii activități sub controlul unui sistem de operare în timp real în care SE este privit ca un task. În acest mod SE devine o parte a pachetului complet de programe în timp real. Evenimentele reactualizate sînt de diferite tipuri și diferă numai după locul în care apar. Din acest punct de vedere se împart în:

- ♦ evenimente externe (exterioare SE) - de regulă datele provenite de la procesul fizic ce sînt obținute în urma tratării întreruperilor;
- ♦ evenimente interne (interioare SE) ce sînt semnale cunoscute de la sistemele de operare însă nu sînt manipulate în paralel dar sînt necesare în execuția taskului asociat sistemului expert. Dintre acestea se menționează:
 - ▶ valorile din baza de cunoștințe;
 - ▶ starea curentă a procesului inferențial;
 - ▶ un semnal de ceas intern.

Toate aceste evenimente sînt asincrone, două sau mai multe dintre ele pot apare în același moment. Sistemul expert trebuie să fie capabil să proceseze datele de intrare cu prioritatea adecvată. Prin tratarea sistemului expert ca un task la sistemele în timp real se vor utiliza facilitățile sistemului de operare în tratarea evenimentelor menționate. Manipularea întreruperilor și activarea taskurilor în funcție de evenimentele apărute este realizată de sistemul de operare.

10.2.4. Implementarea

Sistemul propus în această lucrare este construit din mai multe controllere locale capabile să implementeze algoritmi de conducere PID, conectate prin legătura serială la un calculator compatibil IMB-PC ce îndeplinește și funcția de consola operator de proces. Parametrii de acord pot fi stabiliți fie de la panoul frontal al fiecărui controller, fie de la consola operator care este și mediul de implementare a sistemului expert. Prin legătura serială la nivelul consolei se constituie o bază de date asociată tuturor buclilor de reglare. În această bază de date se găsesc toate informațiile de interes necesare procedurii de acordare optimă. Pentru fiecare controller baza de date asociată are aceeași structură ușurînd astfel accesul. În vederea analizei răspunsului se va genera un buffer circular ce conține valoarea mărimii de ieșire la ultimile N momente de eșantionare. Numărul eșantioanelor este dependent de perioada oscilațiilor procesului. Așa cum s-a prezentat anterior decizia impune

explorarea răspunsului obținut la fiecare set de parametri. Baza de cunoștințe este creată cu ajutorul unui editor de reguli specializat ce crează fișierul text (cunoștințele necesare) și fișierul de reguli (baza de cunoștințe). Reprezentarea prin reguli de producție a fost aleasă atât datorită clarității cât și a ușurinței de implementare. Se dau mai jos exemple de reguli de producție:

```
Rule i.      If parametrul este  $K_p$ 
              (and) Eroarea suprareglaj  $> 0$ 
              [sgn  $W_o E_o = 1$ ]
              Then sgn  $W_o E_o = -1$ 
Rule i + 1.  If parametrul este  $K_p$ 
              (and) Eroare perioada oscilații  $< 0$ 
              Then sgn ( $W_{po} E_{po} = -1$ )
```

Pentru fiecare parametru de acord sînt create reguli specifice pe baza datelor obținute din diagramele prezentate în fig. 10.1. Pe baza acestor reguli este creată baza de cunoștințe cu informații ce au structura

(parametru de acord, mărime de calitate, semn eroare, sens modificare parametru)

Astfel pentru regulile ilustrate mai sus se obține:

(K_p , sigma, 1, -1)

(K_p , P_o , -1, -1) în care,

+1 înseamnă semn pozitiv și -1 semn negativ.

Se dă mai jos sursa C pentru achiziția cunoștințelor referitoare la regulile de producție, a ponderilor asociate criteriilor de performanță și a valorilor impuse.

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
# include <string.h>

# define END 79
# define UP 72
# define HOME 71
# define DOWN 80
# define ENTER 13
# define INSERT 82

void fereastră (int left, int top, int right, int bottom);
void main(void)
{
    int sigma, /* suprareglajul */
        delta, /* raportul suprareglajelor */
        xi, /* dumping-ul */
        po, /* perioada oscilațiilor */
        tc, /* timpul de creștere */
        int w_sigma, /* pondere suprareglaj */
        w_delta, /* pondere raport supra reglaje */
        w_xi, /* pondere dumping */
        w_po, /* pondere perioada de oscilație */
        w_tc, /* pondere timp de creștere */

        int i, j, j_olf, len;
        int left=1, top=1, right=50, bottom=8;
        int left1, top1, right1, bottom1;
```



```

int last, lastx_if, lastx_them last_insert=0;
FILE *parametri, *tabele;
char ch;
char *ptr_begin, *ptr_end;
char *key_word[]={ "sgn",
                    "(",
                    ")",
                    "E_sigma",
                    "w_sigma * E_sigma",
                    "E_delta",
                    "w_delta * E_delta",
                    "E_po",
                    "w_po * E_po",
                    "E_tc",
                    "w_tc * E_tc",
                    "E_xi",
                    "w_xi * E_xi",
                    "= +1",
};
char *title_section[]={ "REGULILE PT. Kc",
                        "Ti",
                        "Td",
}
/* char *param[]={ "sigma"
                  "delta"
                  "po",
                  "tc",
                  "xi"
                  }, */
/* char temp_str1[20], temp_str2[20], temp_str3[20], string1[100],
   string2[100];
temp_str1[0] = 0;
temp_str2[0] = 0;
temp_str3[0] = 0;
string1[0] = 0;
string2[0] = 0;
clrscr();
/* INTRODUCEREA PARAMETRILOR IMPUȘI */
textcolor(WHITE);
left = 1;
top = 1;
right = 50;
bottom = 8;
fereastra(left, top, right, bottom);
window(left, top, right, bottom);
textbackground(RED);
textcolor(WHITE);

gotoxy(2, 2); cprintf ("Introduceți valorile dorite ale
parametrilor:");
textbackground(LICHTBLUE);

gotoxy(2, 3);
cprintf("Suprareglaj; "); cscanf ("%d", &sigma);
gotoxy(20, 3); cprintf("Pondere suprareglaj:");
csacanf("%d", &w_sigma);

gotoxy(2, 4); cprintf("Delta :");
cscanf("%d", &delta);
gotoxy(20, 4); cprintf("Pondere delta :");
csacanf("%d", &w_delta);
gotoxy(2, 5); printf("Perioada :"); cscanf("%d", &po);

```



```

gotoxy(20, 5); cprintf("Pondere perioada :");
cscanf("%d", &w_po);

gotoxy(2, 6); cprintf("Timp crestere:"); cscanf("%d", &tc);
gotoxy(20, 6); cprintf("Pondere t. crestere:");
cscanf("%d", &w_tc);

gotoxy(2, 7); cprintf("Dumping :"); cscanf("%d", &xi);
gotoxy(20, 7); cprintf("Pondere dumping :");
csacanf("%d", &w_xi);

parametri = fopen("PARAM.TXT", "wt");
fprintf(parametri, "%6d%6d\n%6d%6d\n%6d%6d\n%6d%6d\n%6d%6d\n",
        sigma, w_sigma, delta, w_delta, po, w_po, tc, w_tc, xi, w_xi);
tabele=fopen("TABEL.TXT", "wt");
/* CUVINTE DEDICATE */
textbackground(BLACK); textcolor(WHITE);
window(1, 1, 80, 25);
left=52; top=1; right=78; bottom=23;
fereastră(left, top, right, bottom);

window(left, top, right, bottom);
textbackground(RED); textcolor(WHITE);
gotoxy(2, 2); cprintf("CUVINTE DEDICATE");
textbackground(LICHTBLUE);
for(i = 0; i < 15; i++)
{
    gotoxy(2, i + 3);
    cprintf("%s", key_word[i]);
}
gotoxy(2, 3); textbackground(WHITE); textcolor(BLACK);
cprintf("%s", key_word[0]);

window(1, 25, 80, 25); gotoxy(1, 1);
textbackground(WHITE); textcolor(RED);
cprintf("HOME"); gotoxy(5, 1); textcolor(BLACK);
cprintf("Termina regula"); gotoxy(21, 1);
textcolor(RED); cprintf("END"); textcolor(BLACK);
gotoxy(24, 1);
cprintf("Termina grup reguli"); gotoxy(45, 1);
textcolor(RED); cprintf("INSERT"); gotoxy(51, 1);
textcolor(BLACK);
cprintf("Terminare IF"); gotoxy(67, 1); textcolor(RED);
putch(0x18);
textcolor(BLACK); gotoxy(68, 1); cprintf("Up");
gotoxy(72, 1);
textcolor(RED);
putch(0x19);
gotoxy(73, 1);
textcolor(BLACK);
cprintf("Down");

textbackground(LICHTBLUE); textcolor(WHITE);
window(1, 1, 80, 25);
left1=1; top1=10; right1=50; bottom1=23;
fereastră(left1, top1, right1, bottom1);
getch();
for(i = 0; i < 3; i++)
{
    window(left1 + 1, top1 + 1, right1 - 1, bottom1 - 1);
    clrscr();
    gotoxy(2, 2); cprintf("%s", title_section[i]);
    fprintf(parametri, "%s\n", title_section[i]);
    window(1, 1, 80, 25);
}

```



```

j = 1; j_old = 0;
last = 0;
while(1)
{
    if(j > j_old)
    {
        window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
        gotoxy(2, 2); cprintf("Regula%d", j);
        fprintf(parametri, "Regula%d\n", j);
        gotoxy(2, 3); cprintf("IF ");
        cprintf(parametri, "IF "); lastx_if=18;
        gotoxy(2, 5); cprintf("THEN");
        lastx_then=8; gotoxy(8, 3);
        j_old=j;
    }
    chr=getch();
    if(ch==0)
    {
        ch=getch();
        if(ch==END)
        {
            left=52; top=1; right=78; bottom=23;
            window(left, top, right, bottom);
            textbackground(LIGHTBLUE); textcolor(WHITE);
            gotoxy(2, last+3); cprintf("%s", key_word[last]);
            last=0;
            textbackground(WHITE); textcolor(BLACK);
            gotoxy(2, last+3); cprintf("%s", key_word[last]);
            textbackground(LIGHTBLUE); textcolor(WHITE);
            last_insert=0;
            fprintf(parametri, "\n\n");
            break;
        }
        switch(ch)
        {
            case UP:
                left=52; top=1; right=78; bottom=23;
                window(left, top, right, bottom);
                _setcursortype(_NOCURSOR);
                gotoxy(2, last+3);
                textbackground(LIGHTBLUE);
                textcolor(WHITE);
                cprintf("%s", key_word[last]);
                is(last==0) last=14;
                else last--;
                gotoxy(2, last+3);
                textbackground(WHITE); textcolor(BLACK);
                cprintf("%s", key_word[last]);
                textbackground(LIGHTBLUE);
                textcolor(WHITE);
                break;
            case DOWN:
                left=52; top=1; right=78; bottom=23;
                window(left, top, right, bottom);
                _setcursortype(_NOCURSOR);
                gotoxy(2, last+3);
                textbackground(LIGHTBLUE);
                textcolor(WHITE);
                cprintf("%s", key_word[last]);
                is(last==14) last=0;
                else last++;
                gotoxy(2, last+3);

```



```

textbackground(WHITE);
textcolor(BLACK);
cprintf("%s", key_word[last]);
textbackground(LICHTBLUE);
textcolor(WHITE);
break;
case HOME:
gotoxy(8, 7);
cprintf("Totul este corect[y/n]:");
ch=getch();
if(ch=='y')
{
j_old=j;
j++;
left=52; top='; right=78; bottom=23;
window(left, top, right, bottom);
textbackground(LICHTBLUE);
textcolor(WHITE);
gotoxy(2, last+3);
cprintf("%s", key_word[last]);
last=0;
textbackground(WHITE);
textcolor(BLACK);
gotoxy(2, last+3);
cprintf("%s", key_word[last]);
textbackground(LICHTBLUE);
textcolor(WHITE);
window(left1 + 8, top1 + 3, right1 - 2, bottom1 - 2);
clrscr();
last_insert=0;
strcat(string1, "\n");
strcat(string2, "\n");
fprintf(parametri, "%s%s", string1, string2);
ptr_begin=strchr(string1, '_');
ptr_end=strchr(string1, ')');
len=(ptr_end - ptr_begin) - 1;
strncpy(temp_str1, ptr_begin+1, len);
temp_str1[len]=0;
ptr_begin=strchr(string1, '=');
strncpy(temp_str2, ptr, begin+1, 3);
temp_str2[3]=0;
ptr_begin=strchr(string2, '=');
strncpy(temp_str3, ptr_begin+1, 3);
temp_str3[3]=0;
fprintf(tabele, "%4s%8s%4s%4s\n", sect[i], temp_str1,
temp_str2, temp_str3);
string[0]=0; string2[0]=0;
break;
}
else
{
window(left1 + 8, top1 + 3, right1 - 2, bottom1 - 2);
clrscr();
last_insert=0;
window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
gotoxy(2, 2); cprintf("Regula%d", j);
gotoxy(8, 3);
lastx_if=8; lastx_then=8;
string1[0]=0; string2[0]=0;
break;
}
case INSERT:

```



```

        if(last_insert==0)
        {
            last_insert++;
            gotoxy(8, 5);
            strcat(string2, "THEN");
        }
        else
            last_insert=0;
        break;
        default: cprintf("ALTELE"); break;
    }
}
if(ch==ENTER)
{
    if(last_insert==0)
    {
        window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
        _setcursortype(_NORMALCURSOR);
        gotoxy(lastx_if, 3);
        cprintf("%s", key_word[last]);
        strcat(string1, key_word[last]);
        lastx_if+=strlen(key_word[last]);
        gotoxy(lastx_if, 3);
    }
    if(last_insert==1)
    {
        window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
        _setcursortype(_NORMALCURSOR);
        gotoxy(lastx_then, 5); cprintf("%s", key_word[last]);
        strcat(string2, key_word[last]);
        lastx_then+=strlen(key_word[last]);
        gotoxy(lastx_then, 5);
    }
}
}
window(1, 1, 80, 25);
fclose(parametri);
fclose(tabele);
while(!kbhit());
}

void fereastră(int left, int top, int right, int bottom)
{
    int i;
    gotoxy(left, top);
    putchar(0xc9);
    for(i = left + 1; i < right; i++)
    {
        gotoxy(i, top);
        putchar(0xcd);
    }
    putchar(0xbb);
    for(i = top + 1; i < bottom; i++)
    {
        gotoxy(left, i); putchar(0xba);
        gotoxy(right, i); putchar(0xba);
    }
    gotoxy(left, bottom); putchar(0xc8);
    for(i = left + 1; i < right; i++) putchar(0xcd);
    gotoxy(right, bottom); putchar(0xbc);
}

```


Mecanismul de inferență după inspectarea bazei de date determină caracteristicile răspunsului și calculează scorul. Dacă scorul este mai mic decât cel impus în baza de cunoștințe nu se intervine asupra parametrilor de acord. Scorul este calculat ca $S_i W_i E_i$ și indică apropierea față de răspunsul dorit. Valorile noilor parametri de acord sînt obținute pe baza relației (10.7) și setate la nivelul controllerului local. Fiecare iterație crește ordinul șirului a_k ce intervine în relația (10.7). Sursa pentru mecanismul inferențial ce se integrează ca un task este:

```
/* Declarații și definiri de variabile */
void determin_extrem (int far *point, int *er1, int *er2, int
    *er3, int *pozml, int *pozml2, int *pozml3);
void round_max(int far *point, int *ermax, int *pozmax);
void round_next_max(int far *point, int poz_last_max, int *ermax,
    int *pozmax);
void round_mint(int far*point, int poz_last_max, int *ermin, int
    *pozmin);
void modif_suma(int, i, float er, int w, float *suma);
void main(void)
{
    unsigned int far *point_round;
    unsigned int far *point_round;
    unsigned char far *point;
    unsigned char far *point1;

    char nr_bucla;
    float suma;
    int Kc, Ti, Td;
    int Kc_new Ti_new, Td_new, sgn;
    int sigma1, sigma2, sigma3;
    float po_inst, tc_inst, delta_inst, xi_inst;
    int pozmax1, pozmax2, pozmax3, pozmin;

    int sigma_ref;
    float po_ref, tc_ref, delta_ref, xi_ref;
    int w_sigma, w_delta, w_po, w_tc, w_xi;
    float er_sigma;
    float er_delta, er_po, er_tc, er_xi;

    int fibo[] = {2, 3, 5, 8, 13, 21};
    int i, j;
    char car;
    FILE *fp1, *fp2;
    char total_string[100], Kc_string[30], Ti_string[30],
    Td_string[30];
    char ch;
    clrscr();

    /* Comunicația cu bufferul ce conține istoria răspunsului nu este
    dată întrucât este particulară modului de stocare */

    fp1 = fopen("PARAM.TXT", "rt");
    fp2 = fopen("TABEL.TXT", "rt");
    fseek(fp2, 0L, SEEK_SET);
    i = 0; j = 0;
    while((ch = fgetc(fp2)) != 'z')
    {
        buf[i][j] = ch;
        j++;
        if(j%21 == 0)
```



```

{
    j = 0;
    i++;
}
}
determin_extrem (point_round, &sigma1, &sigma2, &sigma3,
                 &pozmax1, &pozmin, &pozmax2);
fseek(fp1, OL, SEEK_SET);
fscanf(fp1, "%6d%6d", &sigma_ref, &w_sigma);
fseek(fp1, OL, SEEK_SET);
fseek(fp1, 13L, SEEK_SET);
fscanf(fp1, "%f%d%c%f%d%c%f%d%c%f%d", &delta_ref, &w_delta, &car,
      &po_ref, &w_po, &car, &tc_ref, &w_tc, &car, &xi_ref, &w_xi);
if(sigma1 > sigma_ref)
{
    er_sigma = (float)(sigma1 - sigma_ref);
    if(sigma2 >= 0)
    {
        sigma2 = 0;
        er_sigma = er_po = er_tc = er_xi = 0;
    }
    else
    {
        po_inst = ((pozmax1 - pozmin) * 2 * T_Esant) / 1000.0;
        tc_inst = (po_inst * PERCENT_Po) / 100.0;
        er_po = po_inst - po_ref;
        er_tc = tc_inst - tc_ref;
        if(sigma3 <= 0)
            er_delta = er_xi = 0;
        else
        {
            delta_inst = ((float)sigma3) / sigma1;
            xi_inst = ((float)(sigma3 - sigma2)) / (sigma1 - sigma2);
            er_delta = delta_inst - delta_ref;
            er_xi = xi_inst - xi_ref;
        }
    }
}
i = 0;
suma = 0;
while(strncmp(buf[i], "Kc", 4) == 0)
{
    if(strncmp(buf[i] + 7, "sigma", 5) == 0)
    {
        modif_suma(i, er_sigma, w_sigma, &suma);
        i++;
    }
    if(strncmp(buf[i] + 7, "delta", 5) == 0)
    {
        modif_suma(i, er_delta, w_delta, &suma);
        i++;
    }
    modif_suma(i, er_delta, w_delta, &suma);
    i++;
}
if(strncmp(buf[i] + 7, "po", 5) == 0)
{
    modif_suma(i, er_po, w_po, &suma);
    i++;
}
if(strncmp(buf[i] + 7, "tc", 5) == 0)
{
    modif_suma(i, er_tc, w_tc, &suma);

```



```

    i++;
  }
  if(strncmp(buf[i] + 7, " xi", 5) == 0)
  {
    modif_suma(i, er_xi, w_xi, &suma);
    i++;
  }
}
if(suma < 0) sgn = -1;
else sgn = 1;
Kc = *((int*)(point + POINTER_Kc));
Kc_new = (int)(Kc * (1 + 0.5 * sgn));
itoa(Kc_new, Kc_string, 10);
suma = 0;
while( strcmp(buf[i], " Ti", 4) == 0)
{
  if(strncmp(buf[i] + 7, "sigma", 5) == 0)
  {
    modif_suma(i, er_sigma, w_sigma, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, "delta", 5) == 0)
  {
    modif_suma(i, er_delta, w_delta, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, " po", 5) == 0)
  {
    modif_suma(i, er_po, w_po, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, " tc", 5) == 0)
  {
    modif_suma(i, er_tc, w_tc, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, " xi", 5) == 0)
  {
    modif_suma(i, er_xi, w_xi, &suma);
    i++;
  }
}
if(suma < 0) sgn = -1;
else sgn = 1;
Ti = *((int*)(point + POINTER_Ti));
Ti_new = (int)(Ti * (1 + 0.5 * sgn));
itoa(Ti_new, Ti_string, 10);
suma = 0;
while( strcmp(buf[i], " Td", 4) == 0)
{
  if(strncmp(buf[i] + 7, "sigma", 5) == 0)
  {
    modif_suma(i, er_sigma, w_sigma, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, " po", 5) == 0)
  {
    modif_suma(i, er_po, w_po, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, " tc", 5) == 0)
  {
    modif_suma(i, er_tc, w_tc, &suma);
    i++;
  }
}

```



```

    }
    if(strncmp(buf[i] + 7, " xi", 5) == 0)
    {
        modif_suma(i, er_xi, w_xi, &suma);
        i++;
    }
}

if(suma < 0) sgn = -1;
else sgn = 1;
Td = *((int*)(point + POINTER_Td));
Td_new = (int)(Td * (1 + 0.5 * sgn));
itoa(Td_new, Td_string, 10);
sprintf(total_string, "NEWKc=%6s NEWTi=%6s NEWTd=%6s",
Kc_string, Ti_string, Td_string);
strcpy((point + POINTER_Buf), total_string);
printf("REZULTATELE EXPERT: \n %s\n", total_string);
}
else
    strcpy((point + POINTER_Buf), "Nu sînt necesare modificări
parametri");
}

void modif_suma(int i, float er, int w, float *suma)
{
    if(er > 0)
    {
        if(buf[i][14] == '+')
        {
            if(buf[i][18] == '-')
                *suma = *suma - w*fabs(er);
            else
                *suma = *suma + w*fabs(er);
        }
    }
    else
    {
        if(buf[i][14] == '-')
        {
            if(buf[i][18] == '-')
                *suma = *suma - w*fabs(er);
            else
                *suma = *suma + w*fabs(er);
        }
    }
}

void determin_extrem (int far *point, int *er1, int *er2, int *er3,
int *poz1, int *poz2, int *poz3)
{
    round_max(point, er1, poz1);
    round_max(point, poz1, er2, poz2);
    round_next_max(point, *poz2, er3, poz3);
}

void round_max(int far *point, int *ermax, int *pozmax)
{
    int poz;
    int poz1;

    poz = *((char*)(point + 60));
    *(ermax) = *((int*)(point + poz));
    *(pozmax) = 0;

    poz1 = poz;

```



```

while(poz1 >= 0)
{
    poz1--;
    if( *(ermax) < *((int*)(point + poz1)) )
    {
        *(ermax) = *((int*)(point + poz1));
        *(pozmax) = poz - poz1;
    }
}
poz1 = 59;
while(poz1 > poz)
{
    if( *(ermax) < *((int*)(point + poz1)) )
    {
        *(ermax) = *((int*)(point + poz1));
        *(pozmax) = 60 - poz1 + poz;
    }
    poz--;
}

void round_next_max(int far *poin, int poz_last_max, int *ermax, int
*pozmax)
{
    int poz;
    int poz1;
    poz = *((char*)(point + 60));
    *(ermax) = *((int*)(point + poz));
    *(pozmax) = 0;
    poz1 = poz;
    while(poz1 >= 0 && poz_last_max > (poz - poz1))
    {
        poz1--;
        if( *(ermax) < *((int*)(point + poz1)) )
        {
            *(ermax) = *((int*)(point + poz1));
            *(pozmax) = poz - poz1;
        }
    }
    poz1 = 59;
    while(poz1 > poz && poz_last_max > (60 - poz1 + poz) )
    {
        if( *(ermax) < *((int*)(point + poz1)) )
        {
            *(ermax) = *((int*)(point + poz1));
            *(pozmax) = 60 - poz1 + poz;
        }
        poz--;
    }
}

void round_min(int far *point, int poz_last_max, int *ermin, int
*pozmin)
{
    int poz;
    int poz1;
    poz = *((char*)(point + 60));
    *(ermin) = *((int*)(point + poz));
    *(pozmin) = 0;
    poz1 = poz;

```



```

while(poz1 >= 0 && poz_last_max > (poz - poz1) )
{
    poz1--;
    if( *(ermin) > *((int*)(point + poz1)) )
    {
        *(ermin) = *((int*)(point + poz1));
        *(pozmin) = poz - poz1;
    }
}

poz1 = 59;
while(poz1 > poz && poz_last_max > (60 - poz1 + poz) )
{
    if( *(ermin) > *((int*)(point + poz1)) )
    {
        *(ermin) = *((int*)(point + poz1));
        *(pozmin) = 60 - poz1 + poz;
    }
    poz1--;
}
}

```

Pentru a se asigura funcționarea on-line partea de inferență a SE este organizată sub formă de task. Sub controlul executivului lucrează și alte taskuri cum este cel de comunicație pentru supravegherea liniei seriale, cel de tastatură și afișare. O funcție task conține tot mecanismul de inferență, funcție care are structura similară cu cea prezentată în paragraful 10.1.

Două aspecte importante au fost avute în vedere: o strategie de ameliorare a parametrilor de acord în funcție de dependența acestora de mărimile de calitate cât și modul de integrare a unui sistem expert într-o aplicație de timp real. Sistemul a fost testat în condiții de laborator, rezultatele obținute fiind promitătoare. În viitor se încearcă perfecționarea prin determinarea și a altor dependente între parametrii de acord și mărimile de calitate.

10.3. Sistem expert pentru diagnoza medicală

Acest exemplu constituie un sistem de diagnosticare în medicină, vizînd detectarea unor maladii comune cum sînt gripă, guturai, viroze. Dezvoltarea de față dispune de o interfață utilizator simplă cu ajutorul căreia medicul răspunde la întrebări privind starea pacientului. Aprecierea stării este judecata prin valorile funcției de apartenență în intervalul $(-5, 5)$. Gradul de adecvanță asociază pentru -5 valoarea absolut false și pentru 5 valoarea absolut true. Valorile cele mai apropiate de 0 sînt cele mai incerte.

10.3.1. Implementarea mecanismului de inferență

```

# include "stdio.h"
# include "stdlib.h"
# include "alloc.h"

```



```

#include "string.h"
#include "math.h"

FILE *fp;
char x[30];
char y[30];
char z[30];
char q[30];
char s1[120];
char s2[120];
char bests[120];
char s3[120];
int i, j, k, ss, best, response, bestvar;
int ipot=0;
int evid=-1;
float *po, *rulevalue, *mini, *maxi;
int *questions, *var, *varflag;
float absolut, p, pp, py, af, pn, pe, rv, maxofmin, prior, a1, a2,
a3, a4;

void main(void)
{
    if((fp=fopen("dataa.dat", "r"))==NULL){
        printf("cannot open file\n");
        exit(1);
    } /*if open*/

    clrscr();
    rewind(fp);
    fscanf(fp, "%s", s1, s1);
    while(strcmp(s1, "symptoms")) { /*whl desc*/
        fscanf(fp, "%s", s, z);
        while(strcmp(z, "999")){
            fscanf(fp, "%s%s", x, y, z);
        }
        ipot++;
        fscanf(fp, "%s", s1);
    }
    while(strcmp(s2, "end")){
        fscanf(fp, "%s", s2, s3);
        evid++;
    } /*0*/

    po=(float *) malloc((ipot + 1) * sizeof(float));
    rulevalue=(float *) malloc((evid + 1) * sizeof(float));
    questions=(int *) malloc((ipot + 1) * sizeof(int));
    mini=(float *) malloc((ipot + 1) * sizeof(float));
    maxi=(float *) malloc((ipot + 1) * sizeof(float));
    var=(int *) malloc((ipot + 1) * sizeof(int));
    varflag=(int *) malloc((evid + 1) * sizeof(int));
    if(!po || !rulevalue || !questions || !mini || !maxi || !var ||
    !varflag){
        printf("allocation failure\n");
        exit(1);
    }
    for(j = 1; j <= (evid + 1); j++){
        varflag[j] = 1;
        rulevalue[j] = 0;
    }
    for(j = 1; j <= (ipot + 1); j++){
        mini[j] = 0;
        maxi[j] = 0;
        po[j] = 0;
        questions[j] = 0;
        var[j] = 0;
    }
}

```



```

    }
    /* find the prior probab and rule values */
    j = 0;
    rewind(fp);
    for (k = 1; k <= ipot; k++) questions[k] = 0;
    fscanf(fp, "%s", s1);
    for(i = 1; i <= ipot; i++){
        fscanf(fp, "%s%s", s1, q, y);
        po[i] = atof(q);
        p = po[i];
        j = atoi(y);
        k = 0;
        while(strcmp(y, "999")){
            fscanf(fp, "%s%s", x, z);
            py=atof(x);
            pn=atof(z);
            k++;
            rulevalue[j]+=fabs(p*py/(p*py + (1 - p)*pn) - p*(1 - py) /
            (p*(1 - py)+(1 - p)*(1 - pn)));
            fscanf(fp, "%s", y);
            j=atoi(y);
        }
        var[i] = k;
        questions[i] = k;
    }
    /* find the maximum symptom and query */
    do {
        rv=0;
        bestvar=0;
        for(j = 1; j <= evid; j++) {
            if (rulevalue[j]>rv) { /*3*/
                bestvar=j;
                rv=rulevalue[j];
            }
            rulevalue[j]=0;
        } /*1*/
        if(bestvar==0) {
            printf("no furthe symptoms\n");
            exit(1);
            fclose(fp);
        }
        fscanf(fp, "%s", s1);
        for(j = 1; j <= bestvar; j++) fscanf(fp, "%s%s", s2, s3);
        printf("\n\nquestions:-\n");
        printf("%s", s3);
        varflag[bestvar] = 0;

        /* obtain user response and adjust probabili */
        printf("\nreply on a scale -5(no) to +5(yes) \n");
        scanf("%d", &response);
        rewind(fp);
        fscanf(fp, "%s", s1);
        for( i = 1; i <= ipot; i++){
            fscanf(fp, "%s%s", s1, q);
            p=atof(q);
            for( k = 1; k <= var[i]; k++) {
                fscanf(fp, "%s%s%s", y, x, z); /*AICI*/
                j=atoi(y);
                py=atof(x);
                pn=atof(z);
                if(j!=bestvar || questions[i]==0) continue;
                questions[i]--;
            }
        }
    } while(1);
}

```



```

    p=po[i];
    pe=p*py + (1 - p)*pn;
    if(response>0) po[i]=p*(1 + (py/pe - 1) *response/5);
    else po[i]=p*(1 + (py - (1 - py)*pe/(1 - pe)) *response/5);
    if(po[i]==floor(po[i])) questions[i]=0;
}
fscanf(fp, "%s", y);
} /*1*/

/* find new rule values and minimum and max probab */
rewind(fp);
fscanf(fp, "%s", s1);
maxofmin=0;
best=0;
for(i = 1; i <= ipot; i++)
    fscanf(fp, "%s%s", s1, q);
    prior=atof(q);
    p=po[i];
    a1=1;
    a2=1;
    a3=1;
    a4=1;
    for(k = 1; k <= var[i]; k++) {
        fscanf(fp, "%s%s%s", y, x, z);
        j=atoi(y);
        py=atof(x);
        pn=atof(z);
        if(varflag[j] *questions[i]==0) continue;
        if(pn>py) { /*4*/
            py = 1 - py;
            pn = 1 - pn;
        }
        rulevalue[j]+=(p*py/(p*py + (1 - p) *pn) - p* (1 - py) / (p*
            (1 - py) + (1 - p)*(1 - pn)));
        a1* = py;
        a2* = pn;
        a3* = (1 - py);
        a4* = (1 - pn);
    } /*includ for k*/
    fscanf(fp, "%s", y);
    maxi[i]=p*a1/(p*a1 + (1 - p)*a2);
    mini[i]=p*a3/(p*a3 + (1 - p)*a4);
    if (maxi[i] < prior) { /*3*/
        questions[i]=0;
        printf("\nwe can ignore %s", s1);
    }
    if(mini[i]>maxofmin) {
        best=i;
        maxofmin=mini[i];
    }
}

/* search for a clear winner */
for(i = 1; i <= ipot; i++) if(mini[best] <= maxi[i] && i!=best)
maxofmin=0;
}
while (maxofmin==0);
rewind(fp);
fscanf(fp, "%s", s1);
for( i = 1; i <= best; i++) {
    fscanf(fp, "%s%s", bests, q);
    af=atof(q);
    for( k = 1; k <= var[i]; k++) {

```



```

        fscanf(fp, "%s%s%s", y, x, z);
    }
    fscanf(fp, "%s", s1);
}
printf("\n\n the most likelu outcome is %s", bests);
printf("\n with a probability %f", po[best]);
fclose(fp);
free(po);
free(rulevalue);
free(questions);
free(mini);
free(maxi);
free(var);
free(varflag);
}

```

10.3.2. Baza de cunoștințe

Baza de cunoștințe este realizată sub forma unui fișier de date ce grupează simboluri pentru diferite simptome. Structura sa este dată mai jos.

illenesses

common_cold

1.0	.01	7
1.0	.01	2
1.0	.01	6
1.0	.01	16
1	.01	7
1.0	.01	16
0.0	.5	8
0.0	.01	64
1.0	.01	15
1.0	.01	17
0.0	.01	999 laryngitis
1.0	.01	8
1.0	.01	34
1.0	.01	12
1.0	.01	15
1.0	.01	18
1.0	.01	31
1.0	.01	22
1.0	.01	12
1.0	.01	22
1.0	.01	36
1.0	.01	23
1.0	.01	5
1.0	.01	5
1.0	.01	18
1.0	.01	23
1.0	.01	29
1.0	.9	7
1.0	.01	18
1.0	.01	999

//Sînt date mai jos o serie de simptome

symptoms

1

are-you-snezing-a-lot?

- 2 are-your-eyes-painful-or-waterin-a-lot?
- 3 do-you-have-a-sore-throat?
- 4 is-your-voice-hoarse?
- 5 are-you-coughing-a-lot?
- 6 do-you-have-a-runny-nose?
- 7 do-you-have-a-headache-or-in-general-do-you-sufer-from-headaches-at-all?
- 8 do-you-have-a-high-temperature?
- 9 do-you-spend-a-lot-of-your-time-in-a-very-dusty-atmosphere?
- 10 dose-your-skin-itch?
- 11 do-you-have-a-dry-throat?
- 12 is-your-breath-"wheezy"?
- 13 is-your-nose-very-'blocked-up'?
- 14 have-you-had-a-cold-or-similar-infection-recently?
- 15 do-you-feel-generally-ill?
- 16 do-you-have-trouble-swallowing?
- 17 do-you-muscles-ache?
- 18 do-you-have-any-pain-at-all-in-your-chest?
- 19 have-you-had-your-tonsils-removed?
- 20 do-you-have-any-symptoms-which-tend-to-occur-in-'attacks-'rather-than-being-present-all-the-time?
- 21 do-you-have-a-'productive-'cough-a-cough-in-which-you-bring-something-up?
- 22 are-you-rather-breathelss?
- 23 do-you-sweat-a-lot-not-just-when-you-exert-yourself-but-when-you-are-apparently-relaxing-as-well?
- 24 is-your-pulse-rate-high-nomally-it-should-be-about-60-to-80-beats-each-minute-and-slight-faster-for-people-over-70-or-under-20.
- 25 do-you-have-severe-attacks-of-breathlessness-enough-to-seriously-worry-you?
- 26 does-your-skin-have-a-bluish-tinge?
- 27 when-you-cough-is-your-phlegm-stained-with-blood?
- 28 are-you-confused-muddled-about-what's-going-on-around-you?
- 29 are-you-(or-the-patient)-delirious-talking-incoherently-with-poor

- muscular-coordination?
- 30
do-you-have-a-dry-(non-productive)-cough?
- 31
is-it-painful-when-you-breathe-or-cough?
- 32
do-you-ever-have-any-really-severe-pain-in-your-chest?
- 33
do-you-swing-between-feeling-chilled-and-feeling-feverish?
- 34
do-you-have-any-symptoms-wich-have-been-present-for-some-time-possibly-six-weeks-or-more?
- 35
do-you-have-'clubbed-fingers'?-these-are-fingers-in-wich-the-cuticles-have-almost-disappeared-and-nails-curve-over-at-the-fingertips?
- 36
do-you-have-any-symptoms-wich-mainly-occur-when-you-exert-yourself?
- 37
do-you-smoke?-to-reply-divide-the-number-of-cigarettes-you-smoke-by-5-use-this-number-to-reply-(if-you-smoke-20-a-day-reply-4).5-is-the-maximum-reply-and--5-means-you-do-not-smoke.
- 38
do-you-suffer-from-feelings-of-dizziness?
- 39
do-you-have-palpitations?-the-feeling-that-your-heart-is-beating-more-strongly-or-faster-or-more-unevenly-than-it-should?
- 40
is-either-your-ankles-unduly-swollen?
- 41
are-you-vomiting-or-do-you-have-strong-feelings-of-nausea?
- 42
do-you-have-any-abdominal-pain? this-is-pain-anywhere-between-the-bottom-of-the-tibcage-and-the-groin.
- 43
do-you-suffer-from-disaphoea?-passing-unusually-runny-feeces.
- 44
have-you-your-appendix-removed?
- 45
do-you-have-jaundice? this-is-not-a-disease-but-a-simptom-of-disease, often-it-is-most-obvious-in-the-eyes-the-whites-become-yellow.
- 46
are-you-rather-tense-and-apprehensive?
- 47
do-you-find-it-hard-to-get-to-sleep-or-do-you-often-wake-in-the-middle-of-the-night?
- 48
do-you-have-any-invluntaru-twitching-or-trembling?
- 49
do-you-suffer-from-constipation?passing-faeces-infrequently-or-with-difficulty.
- 50
do-you-have-a-poor-memory? that-is-difficulty-remmebering-individual-facts-either-occasionally-or-regularly.
- 51
have-you-totally-or-nearly-lost-the-power-of-speech?
- 52
have-you-experienced-any-bleeding-from-your-back-passage?
- 53
are-you-male-or--female?answer-5-for-male-or--5-for-female.if-you'd-like--the-analysis-to-be-general(for-either-sex)-reply-0.
- 54

is-your--neck-stiff-and/or-painful?
 55
 have-you-sustained-any-kind-of-had-injury-over-the-last-few-weeks?-
 even-a-very-slight-injury-can-be-importan.
 56
 have-you-recently-been-passing-abnormal-looking-faeces-?
 57
 are-you-passing-large-quatities-wind-either-by-belching-orflatulence?
 58
 do-you-have-sudden-feeling-of-faintness-feeling-weak-and-unsteady-
 maybe-even-losing-counciousness?
 59
 does-any-part-of-your-body-itch-with-or-without-the-presence-of-any-
 rash?
 60
 do-you-have-a-skin-rash-of-any-sort?
 61
 is-any-part-of-your-numb-or-do-you-have-tingling-'pins-and-
 needles'sensation-anywhere?
 62
 are-you-overweight-or-underweight? reply-5-for-definite-overweight-
 and-5-for-definite-underweight.reply-0-if-your-weight-is-just-right.
 63
 do-you-have-any-pain-in-your-face-or-forehead?
 64
 do-you-have-any-swelling-under-the-skin?
 65
 is-your-urine-abnormally-coloured?
 66
 are-you-urinating-unusually-frequently-?
 67
 is-it-painful-when-you-urinate?
 68
 is-your-vision-impaired-in-any-way-blurring-or-double-vision-
 orseeing-flashing-lights? (this-does-not-include-defects-wich-canbe-
 corrected-by-spectacles)
 end
 end

BIBLIOGRAFIE

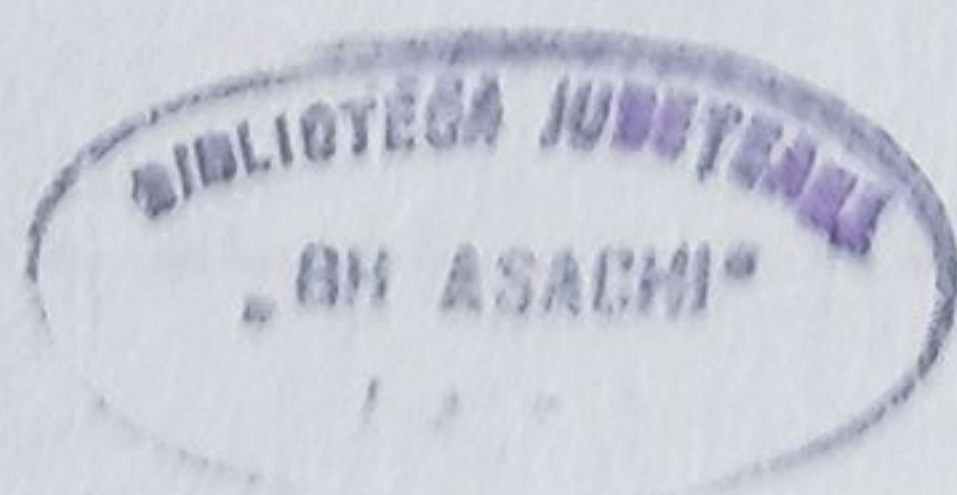
- [1] **Fuzzy decision trees.** In: **Fuzzy sets and systems** - *Adamo, J.M.* 4(1980), pp. 207-219.
- [2] **Introduction a la conception des systems Intelligents** - *Alexander, J.*, Paris, Massen, 1984.
- [3] **Expert Systems Concepts and Examples** - *Alty, J.L., Coombs, M.J.*, N.C.C. Publications, England, 1984.
- [4] **Use of expert systems in closed loop feedback control in Proc.** *Arzer, K.E.*, 1986; Amer. Control Conf. Boston M.A. June 1921, 1985.
- [5] **Comparison of fuzzy sets on the same decision space.** In: **Fuzzy sets and systems** - *Baldwin, J.F., Guild, N.C.F.* 2(1979), pp. 213-232.
- [6] **Pattern recongnition with fuzzy objective function algorithms** - *Bezdec, James C.* - . New York, Plenum Press, 1981.
- [7] **Integration of an Expert System into a Real-Time Software System** - *Beck, Th., Lauber, R.J.* , 1989.
- [8] **Expert System Applications** - *Bolc, L., Coombs, M.J.*, Springer-Verlag, 1988.
- [9] **L'intelligence artificielle** - *Bonnet, A.*, InterEditions, 1984.
- [10] **A computer based consultant for mineral exploration.** *Buda, R. Hart, P.* - Final Report, 1979.
- [11] **Implementations of Prolog** - *Campbell, J.A.* New York, John Wiley, 1985.
- [12] **L'Anatomie de prolog** - *Caneghem M.V.* , InterEditions, 1986.
- [13] **Sisteme de testare a sistemelor cu procesoare de 8 biti.** - *Cârstoiu, D.*, Conferinta de microprocesoare, CNETAC, București, 1986.
- [14] **Sistem expert de autodiagnoză a modulelor multiprocesor.** *Cârstoiu, D.* Sesiunea IRNE Pitești, 1988.
- [15] **Contribuții la optimizarea proceselor de producție utilizând sisteme expert în industria chimică** - *Cârstoiu, D.* Teza de doctorat, Institutul Politehnic București, iulie, 1989.

- [16] **Inference Deficiencies in Rule - Based Expert Systems.** - Cârstoiu, D., 8 - th International Conference on Control Systems and Computer Science, june 1991.
- [17] **Asistarea deciziilor clinice prin calculator** - Cârstoiu, D., Revista Academică, București, nr. 4, 1991.
- [18] **Multi-port Communication between Computers through Radio Connection** - Cârstoiu, D., Radu, C., . Revue Roumaine des Sciences Technique. Serie Electronique et Energetique, 37, 4, p. 523-530, 1992.
- [19] **An Expert Sistem for Improved on line Controller Performance** - Cârstoiu, D., Radu C., . Revue Roumaine des Sciences Technique. Serie Electronique et Energetique, 37, 1, 1993.
- [20] **On-line Expert System to Increase the Quality of PID Controllers. IMACS/IFAC Mathematical and Inteligent Models in System Simulation**, Cârstoiu, D., Radu, C., Brussels, Belgium, 1993.
- [21] **Parallel Inference Machine Architecture for Rule Based Systems** - Cârstoiu, D., . 9-th International Conference on Control Systems and Computer Science, may 1993.
- [22] **Aspects of Physical Database Design and Reorganisation** - Cârstoiu, D., . 9-th International Conference on Control Systems and Computer Science, may 1993.
- [23] **Strategii de testare în sistem cu microprocesor.** Cârstoiu, D., Sandor, V., Algiu, M., Conferința de microprocesoare, microcalculatoare și aplicații, București, noiembrie, 1987.
- [24] **Sisteme de gestiune a modelelor sistemelor automate** - Cârstoiu, D., . A 4-a Conferința Națională de Electronica, Telecomunicații, Automatică și Calculatoare, București, decembrie, 1988.
- [25] **Arhitecturi specifice sistemelor expert cu aplicații în conducerea proceselor. Structuri, algoritmi și echipamente de conducere a proceselor industriale** - Cârstoiu, D., , Iași, p. 333-339, octombrie, 1989.
- [26] **Sisteme expert pentru asistarea deciziilor, Structuri, algoritmi și echipamente de conducere a proceselor industriale** - Cârstoiu, D., , Iași, p. 373-379, octombrie, 1989.
- [27] **Sisteme expert pentru prelucrarea bazelor de date, Structuri, algoritmi și echipamente de automatizare a proceselor industriale** - Cârstoiu, D., , Craiova, martie 1991.
- [28] **Sisteme suport de decizie cu aplicații în medicină. Structuri, algoritmi și echipamente de conducere a proceselor industriale**, Cârstoiu, D., Iași, p. 101-109, octombrie, 1991.
- [29] **Sisteme expert pentru supravegherea calității conducerii.** Cârstoiu, D., SIMSID 7, Galați, p. 261-267, România, 1992.

- [30] **Sisteme expert de autodiagnoză în structuri multiprocesor** - Cârstoiu, D., Sesiunea Jubiliară 175 ani de învățămînt, Iași, noiembrie 1987.
- [31] **Expert Systems in Real Time Control of Technological Processes.** Cârstoiu, D., TITAN '89 Section on Institute for Computers and informatics, Predeal, October, 1989.
- [32] **Systemes experts methodes et outils**, Eyrolles, Chatain, J.N., Dussauchoy, A. 1987.
- [33] **Programming in Prolog** - Clocksin, W.F. Mellish, C.S., Springer-Verlag, 1984.
- [34] **Explaining induced decision trees.** Proceedings Expert Systems - Corlett, R.A., 1983.
- [35] **Sistemi informativi intelligenti** - Davies, R. . Padova, France Muzzio Editore, 1988.
- [36] **An Expert PID controller in Proc.**, Devarathan, R., IEEE Int. Workshop on Artificial Intelligence for Industrial Applications, 1988.
- [37] **Tinning Maps for Three Mode Controllers** - Dorris M. Wills , 1962.
- [38] **Intelligence artificielle: mythes et limites** - Dreyfus, H. Paris, Flammarion, 1979.
- [39] **A logic programming metalanguage for expert systems** - Ernst, CH., J., In: Kacprzyk, J., Yager, R. (eds), 1985, pp. 280-288.
- [40] **Les SE: principes d'organisation et fonctionnement. Informatique et Gestion** - Farreny, H. -, no. 142, mai 1983.
- [41] **A general model to troubleshooting and its applications to computer support.** - Frley, A. 5es journees internationales: les SE - leurs applications, Avignon, 1985.
- [42] **La cinquieme generation.** Feigenbaum, E., McCorduck, P. - Paris, Inter Editions, 1984.
- [43] **Elemente de inteligență artificială.** Georgescu, I., Academia Republicii Socialiste România, București, 1985.
- [44] **PROLOG** - Giannesini, F., Paris, Masson, 1983.
- [45] **Uncertainly modele for knowledge based systems. A unified approach to the measurement of uncertainly.** - Goodman, J.P., Nguyen, N.T. Amsterdam, Elsevier, 1985.
- [46] **Handbook, of Algorithms and Data Structures** - Gonnet, G.H., Addison-Wesley, 1984.
- [47] **Techniques algorithmiques pour l'intelligence artificielle**, Griffiths, M. Paris, Hermes, 1986.
- [48] **Aproximative reasoning in expert systems**, Gupta, M., Kandel, A., s. a, Amsterdam, 1985.

- [49] **Experience in the Use of an Inductive System in Knowledge Engineering.** Proceedings of the Fourth Technical Conference of the British Computer Society Group on Expert Systems, *Hart, A. E.*, 1985.
- [50] **Building Expert Systems**, *Hayes, R., Waterman, D.A., Lenat, D.B.*, Addison-Wesley, 1983.
- [51] **Intelligent decision support in process environments**, *Hollinagel, E.*, Springer-Verlag, 1986.
- [52] **Highly Versatile in Circuit Emulator**, *Ionescu, T.D., Cârstoiu, A.* . 7-th International Conference on Control Systems and Computer Science, may 1987.
- [53] **Teoria sistemelor**, *Ionescu, V.*, , Editura Didactică și Pedagogică, București, 1985.
- [54] **Theory of fuzzy subsets**, *Kaulmann, A.*, Academic Press, New York, 1965.
- [55] **Prolog for programmers**, *Kluzniak, F. Szpakowicz, S.*, Academic Press, 1985.
- [56] **Knowledge representation from a knowledge engineering point of view**, *Konrad, W.*, Proceedings of the International Workshop Spa, Belgium, Springer-Verlag, 1986.
- [57] **"Self-tuning PID controller uses pattern recognition approach**, *Kraus, T.W. and T.J. Myron*, Control eng. pag 106-111 June 1984.
- [58] **Qualitative Process Theory**, *Kuipers, B.* Artificial Intelligence Journal, pp. 85-168, North-Holland, 1984.
- [59] **An artificial intelligence approach to discovery in Mathematics as heuristic search**, *Lenat, D.*, Memo AIM-206, Stanford AI Laboratory, 1976.
- [60] **All about chess and computers**, *Levi, D.*, Berlin, Springer-Verlag, 1982.
- [61] **Applications of Artificial Intelligence for organic chemistry**, *Lindsay, R.K.*, New York, 1980.
- [62] **An Expert System to Perform On-Line Controller Tuning**, *Litt, J.*, American Control Conference, May, 22-25, 1990.
- [63] **A theory of syntactic recognition for natural language**, *Marcus, M.P.*, Cambridge, 1980.
- [64] **A Knowledge Based Environment for Process Planning**, *Mouleswarn, C.B., Fischer, H.*, Proceedings of the 1-st International Conference Southaton, Springer-Verlag, 1986.

- [65] **Expert Systems and Fuzzy Systems**, Negoita C.V., The Benjamin Cummings Publishing Company Inc., 1985.
- [66] **Concepts of expert systems and languages for their implementation**, Petzsch, H., Proceedings of the International Workshop spa, Belgium, Springer-Verlag, 1986.
- [67] **Expert Systems in Engineering**, Pham, D.T., Springer-Verlag, 1988.
- [68] **Real Time expert tunners for PI controllers**, Porter, B., Jones, A. and C.B. McKeown. IEEE Pric. D. vol 134, pp. 260-263 July 1987.
- [69] **Discovering rules by introduction from large collections of examples**, Quinlan, J.R., In Expert in the Microelectronic Age, Edinburgh, 1979.
- [70] **An expert system using palusible inference**, Reit, J., AL/X: Oxford: Inteligent, Terminals Ltd, 1980.
- [71] **Natural Language inferences**, Rich, E., Auckland, Mc-Graw-Hill, 1984.
- [72] **Sisteme expert de autodiagnoză în structuri cu mai multe procesoare**, V. Sandor, D. Cârstoiu, M. Algiu, TESS. Revista Automatica, Electronica, Electrotehnica no. 2/1989.
- [73] **Process Control Systems**, Shirskey, F.G., - New York, McGraw-Hill, 1988.
- [74] **"Applied Optimal Control", Large Scale Systems**, Singh, M.G., Recenzia lucrării 1981.
- [75] **Data Structure Techniques**, Standish, T.A., Addison-Wesley, 1980.
- [76] **Microprocessors in Signal Processing, Measurement and Control**, Tzafestas, K. - Dordrecht D. Reidel, 1983.
- [77] **Optimum settings for automatic controllers**, Ziegler, J.G. and N.B. Nichols - Trans A.S.M.E. vol 64, pp. 759-768, 1942.
- [78] **Fuzzy sets in Decision Making and Expert Systems**, Zimmermann, H.J., Boston, 1987.
- [79] **A Guide to Controller Tuning**, Wills, M., IEEE Transaction on Industrial Electronics, vol. 37, oct. 1990.
- [80] **Comments on "Optimal gaim for proportional-integral derivative feedback"**, Wolbrot, M. and M.P. Polis - IEEE Cotrol Syst. Mag. vol. 9, pp. 100-101, June, 1989.
- [81] **An Expert System for Real-Time Control IEEE Software**, Wright I., 1986, March, pp. 16-24.



Autorul abordează din diferite unghiuri vastul domeniu al Sistemelor Expert, reușind astfel să supună atenției cititorului o problemă pe cât de complexă pe atât de unitară. Lucrarea surprinde unele fundamente ale Teoriei Sistemelor Expert, dar și câteva tendințe de ultimă oră în acest domeniu, venind astfel deopotrivă în sprijinul celor care doresc să se inițieze, cât și celor avansați în domeniul inteligenței artificiale.

Diversificata problemă abordată, maniera riguroasă de prezentare și caracterul unitar impun prezenta lucrare drept una de referință în peisajul publicistic de profil. O recomandăm studenților, informaticienilor, medicilor, inginerilor.

ISBN 973-9156-92-1

8950 lei